2021-06

USING CONVOLUTION NEURAL NETWORKS TO DEVELOP ROBUST COMBAT BEHAVIORS THROUGH REINFORCEMENT LEARNING

Cannon, Christopher T.; Goericke, Stefan

Monterey, CA; Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

USING CONVOLUTION NEURAL NETWORKS TO DEVELOP ROBUST COMBAT BEHAVIORS THROUGH REINFORCEMENT LEARNING

by

Christopher T. Cannon and Stefan Goericke

June 2021

Thesis Advisor:	Christian J. Darken
Second Reader:	Sean A. Clement

Research for this thesis was performed at the MOVES Institute.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 2021	3. REPORT TYPE AND DATES COVERED Master's thesis

4. TITLE AND SUBTITLE
USING CONVOLUTION NEURAL NETWORKS TO DEVELOP ROBUST COMBAT BEHAVIORS THROUGH REINFORCEMENT LEARNING

5. FUNDING NUMBERS

6. AUTHOR(S) Christopher T. Cannon and Stefan Goericke

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

The success of reinforcement learning (RL), as shown with video games such as StarCraft and DOTA 2 achieving above-human performance levels, begs questions about the future role of the technology in military constructive simulations. The objective of this study was to use convolutional neural networks (CNN) to develop artificial intelligence (AI) agents capable of learning optimal behaviors in simple scenarios featuring multiple unit and terrain types. This thesis sought to incorporate a multi-agent training regimen that could be employed in the domain of military constructive simulations. Eight different scenarios, all with varying levels of complexity, were used to train agents capable of exhibiting multiple types of combat behaviors. Overall, the results demonstrate that the AI agents can learn robust tactical behaviors required to achieve optimal or near-optimal performance in each scenario. The findings additionally indicate that a better understanding of multi-agent training was attained. Ultimately, CNN combined with RL techniques prove to be an efficient and viable method to train intelligent agents in military constructive simulations, and their application can potentially save human resources in the execution of live exercises and missions. It is recommended that future work should investigate how to best incorporate similar deep-RL methods into an existing military program of record constructive simulation.

14. SUBJECT TERMS
artificial intelligence, neural network, machine learning, constructive simulation, combat behaviors, reinforcement learning, RL, convolutional neural networks, CNN, artificial intelligence, AI

15. NUMBER OF PAGES
131

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

USING CONVOLUTION NEURAL NETWORKS TO DEVELOP ROBUST COMBAT BEHAVIORS THROUGH REINFORCEMENT LEARNING

Christopher T. Cannon
Captain, United States Marine Corps
BS, U.S. Naval Academy, 2015

Stefan Goericke
Major, German Army
B.Sc. in Economics, Helmut-Schmidt-University, 2009
M.Sc. in Economics, Helmut-Schmidt-University, 2011
B.Sc. in Business Information Systems, University of Applied Science Wismar, 2016

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MODELING, VIRTUAL ENVIRONMENTS, AND SIMULATION

from the

NAVAL POSTGRADUATE SCHOOL
June 2021

Approved by: Christian J. Darken
Advisor

Sean A. Clement
Second Reader

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The success of reinforcement learning (RL), as shown with video games such as StarCraft and DOTA 2 achieving above-human performance levels, begs questions about the future role of the technology in military constructive simulations. The objective of this study was to use convolutional neural networks (CNN) to develop artificial intelligence (AI) agents capable of learning optimal behaviors in simple scenarios featuring multiple unit and terrain types. This thesis sought to incorporate a multi-agent training regimen that could be employed in the domain of military constructive simulations. Eight different scenarios, all with varying levels of complexity, were used to train agents capable of exhibiting multiple types of combat behaviors. Overall, the results demonstrate that the AI agents can learn robust tactical behaviors required to achieve optimal or near-optimal performance in each scenario. The findings additionally indicate that a better understanding of multi-agent training was attained. Ultimately, CNN combined with RL techniques prove to be an efficient and viable method to train intelligent agents in military constructive simulations, and their application can potentially save human resources in the execution of live exercises and missions. It is recommended that future work should investigate how to best incorporate similar deep-RL methods into an existing military program of record constructive simulation.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AI	Artificial Intelligence
C2	Command and Control
CNN	Convolutional Neural Network
DOD	Department of Defense
DotA2	Defense of the Ancients 2
DQL	deep Q-Learning
LSTM	long-short-term memory network (LSTM)
MAGTF	Marine Air-Ground Task Force
MCTS	Monte-Carlo Tree Search
ML	Machine Learning
MLP	Multi-Layer-Perceptron
MTWS	MAGTF Tactical Warfare Simulation
OneSAF	One Semi-Automated Force
PPO	Proximal Policy Optimization
PK-DQN	Prior Knowledge-Deep Q Network
ReLU	rectified linear activation function
RL	Reinforcement Learning
RTS	real-time strategy game
SVG	Scalable Vector Graphics
TRPO	Trust Region Policy Optimization
VPG	vanilla policy gradient

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

We would like to thank Dr. Chris Darken for his unwavering support and guidance throughout our time at NPS. We are incredibly grateful for the countless hours of professional guidance you graciously offered on a weekly basis to help complete this thesis. We would also like to thank Major Sean Clement and TRAC-Monterey for the substantial amount of knowledge, resources, and advice provided throughout our research. Additionally, we would like to thank Mr. Chris Fitzpatrick for opening up the MOVES laboratory and allowing us to use every single machine on a daily basis to train each of our agents. The number of scenarios explored would not have been possible without his assistance. Finally, we would like to thank the best MOVES cohort ever for making our experience at NPS truly unforgettable.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. COMBAT MODELING AND WARFARE

The desire to correctly anticipate an opponent's strategic or tactical behavior in war is as old as humanity's ability to fight these wars [1]. In ancient China, games like Wei Hei and Go were initially used as a method to strengthen military and political leaders' strategic thinking abilities. Later the Romans utilized sandboxes to discuss own and enemy's possible moves before a campaign or a battle. However, it was not before the Prussians with their *Kriegsspiel* (War Game) in the early 19th century that games with a strict set of rules became utilized to predict possible outcomes of military engagements. While these wargames became increasingly popular over the next decades in many armed forces around the world, a limited capacity to conduct necessary calculations always restricted the level of complexity these board-based wargames were able to achieve. Additionally, the physical limitations of board-games restricted the designers to simplified behaviors and game elements rather than striving for realism. However, improved computing power and user-friendly graphical interfaces enabled designers in the late 20th century to model wargames in a higher complexity regarding rules and the number of components in the games. Furthermore, computers' utilization allowed the implementation of computer-based adversaries to play successfully against a human player based on a hard-coded rule-based AI-software.

Today, computer-based wargames, also referred to as constructive simulations [2], have become a useful tool across the Department of Defense (DOD). They allow military leaders to further learn and develop their operational procedures in areas that are often deemed too costly or dangerous to rehearse regularly. Leaders are provided the ability to employ their forces against multiple red force designs before any live execution, yielding them an opportunity to validate their scheme of maneuver without assuming any additional risk. Large unit staffs at the strategic level often resort to using constructive simulations as the method of training [3], where leaders can make inputs inside of a simulated environment, but they are not involved in determining the outcomes of scenarios [2].

B. SCOPE AND PROBLEM STATEMENT

The method used to represent adversarial behaviors in computer-based wargames requires either direct encoding from the scenario designer through scripts or using live human players for all red force decision making. Both methods provide a sufficient resolution to represent adversarial behaviors, but each come with their disadvantages [4]. Directly encoding specific behaviors may be possible for low-level scenarios, but as scenarios are scaled upward, the number of units and possible actions become too challenging for a script to control and often result in unrealistic behaviors [4]. For the larger scenarios, using human players as the red force may provide more realistic results, but the additional human resources cause a logistical strain, and the overall productivity is limited on the knowledge and ability of the individual players.

A possible approach to solve this problem may lie in the utilization of artificial neural networks. In the domain of computer gaming, this approach has lately proven to be quite successful. For instance, for the real-time strategy game StarCraft II, an artificial neural network was developed that defeated 99.8% of the players regularly participating in online competitions [5]. While in the domain of computer games, the utilization of artificial neural networks has made vast progress recently, but within the domain of military used wargames, research is only beginning. In recent research, Boron [6] and Sun et al. [7]. have shown that artificial neural networks are suitable to solve challenges in simple military wargame scenarios. Based on the previous work, especially that conducted by Boron [6], this thesis aims to increase the complexity within the used military scenarios. While Boron used a simple Multi-Layer-Perceptron (MLP) neural network, this architecture turned out to be unsuitable when dealing with dynamical starting positions of own and enemy units as well as dynamic enemies' behavior. Moreover, the used scenarios were limited to a maximum of five units on the battlefield [6]. In this thesis, a training simulation will be built to support a Convolutional Neural Network (CNN) architecture and include multiple units and terrain types to overcome these restrictions. Additionally, multi-agent training will be applied in a defined scenario to test if this approach can successfully be utilized in the domain of military constructive simulations.

C. BENEFIT AND STRUCTURE OF THESIS

Due to the topic selection and aforementioned restrictions described, this thesis is more likely to be assigned to basic research within the field of military constructive simulations. A quantifiable benefit, such as savings of personnel and material, will not be achieved at the end of the work. Nevertheless, the work will deepen the understanding of the uncertainties and difficulties of utilizing neural networks and multi-agent learning to simulate own and enemies' unit's behavior. In the long run, it will contribute to saving human resources in the preparation and execution of exercises and missions.

The first chapter of this thesis serves as a general introduction to the topic. The problem is narrowed down, and the research question is formulated. In the second chapter, underlying concepts such as machine learning, reinforcement learning (RL), and neural networks will be explained. Additionally, it will include a general overview of artificial intelligence (AI) applications within the military domain, including an overview of the current state of research within the domain of military constructive simulations. In the third chapter, the used software framework will be described. In the fourth chapter, the different used scenarios will be explained, and the results will be laid down. Thereby, the discussion of the results for each experiment will be included. In the fifth chapter, a general conclusion based on Chapters II and IV will be given.

THIS PAGE INTENTIONALLY LEFT BLANK

II. CONCEPTS, EXAMPLES AND CURRENT RESEARCH

In the last chapter, the goal, scope, and structure of this thesis was presented. This chapter aims to explain the theoretical background used for this work. In the first part, basic concepts will be explained. After that, an overview of the current state of research in the field of neural networks-based AI within the game industry and military applications are given.

A. CONCEPTS

In this sub-chapter, basic concepts such as machine learning, reinforcement learning, or neural network will be explained. The sub-chapter starts with an overview of the different kinds of simulations used within DOD.

1. Types of Simulations

With the exponential growth of computer technology and the improvements made within the field of input and output devices, simulations of all kinds have become an important part in preparing military forces for a battle. Simulations not only allow military leaders to train their forces on all operating levels, but also let them assess and improve current schemes of maneuver and tactics. The simulations utilized throughout the DOD can be categorized by the method used to represent people and the environment that they are interacting with. These categories include live, virtual, and constructive.

In virtual simulations, real people operate simulated systems. A widely known form of such kind of simulation is an aircraft simulator. The main purpose of such simulation systems is to train individuals or small groups in different skills such as operating a specific piece of equipment, making proper decisions in prior defined situations, or communicating in accordance with given procedures and regulations. Live simulations, on the other-hand, consist of actual people operating with their real equipment [2] in a real environment. Live simulations are used to rehearse combat drills against simulated or notional enemy in warlike conditions. While often effective in disseminating instant feedback to the individual Soldier, live simulations come with several limiting factors. To best replicate a

combat environment, live simulations require the participating units and equipment to operate at strenuous conditions for extended periods. The deterioration of the equipment when operated at this level requires a significant amount of maintenance, which is often costly and time-consuming [3]. Live simulations are often limited to the amount of available space that is dedicated toward training. In the United States, there are only a few training locations that facilitate live simulation training for units more extensive than a battalion. With this limitation, live simulations prohibit senior leadership's opportunity at the strategic and operational level to train and exercise precise Command and Control (C2).

To overcome this shortage, or when the training objective is to solely enhance the senior leaderships' ability in conducting C2, the DOD often resorts to using constructive simulations as their alternative training tool. In constructive simulations, both the people and the environment they interact with are simulated [2]. Since constructive simulations are typically used on higher levels of command, units and terrain can be represented at a much larger scale through modeling an environment with lower fidelity. In constructive simulations, mathematical models and algorithms represent units, weapons, equipment, attrition rates, and the combat behaviors portrayed by each force operating in the designated environment. Commanders then observe the interactions between the two forces and analyze the tactics demonstrated by each force. A commander's ability to interact with the constructive simulation environment can be categorized as an open form or a closed form [8]. A closed-form simulation requires input from the commander initially, but as the scenario is executed, all other decisions for each of the forces are made using scripted behaviors. Comparatively, in open form simulations, both forces are allowed to make inputs throughout the entire scenario and base their future behaviors on how the scenario unfolds before them. The term wargame is widely used as a synonym for military constructive simulations.

2. Machine Learning and Deep Reinforcement Learning

Machine learning (ML) is a subdivision of AI that uses computer algorithms to train an agent that uses its past experiences to improve its performance [9] rather than explicit programming to improve its performance. Based on the method that the learning algorithm is

provided information or feedback, machine learning can be typically broken down into three categories: supervised learning, unsupervised learning, and reinforcement learning [10].

In supervised learning, an agent is presented with a data set where inputs are always associated with an output that is known to be correct [10]. The agent uses that information to learn and create a function with rules to always associate those inputs with those outputs. This method can also be explained as the teacher method. For instance, imagine students in elementary school are learning how to identify different types of animals correctly. A teacher might start by showing the students several pictures that explicitly state that the animal in the image is a dog. The students will then associate common regularities found in each picture, such as four legs, tail, pointy nose, and fur, to be that of a dog. The teacher will then test the students by showing them new images that they have never seen before, where they will be expected to identify all images that contain a dog correctly. If the students can correctly identify all of the images with dogs, learning can be deemed sufficient.

Unsupervised learning on the other hand is substantially different. For unsupervised learning, the agent looks to associate inputs with previously undetected patterns, even though no known information or labels are provided [10]. Expanding on the previous analogy, picture again students in elementary school, but this time, they learn how to associate animals with each other, but they are not learning what the animals are. The students are once again provided with pictures of animals, except this time, they do not have anything labeling the animal's identity. The students are then told to look for commonalities between the images and separate them into different piles however they saw fit. If the students could separate the images into separate piles containing the same type of animal, then training was practical. The students may not know that the picture of a dog is actually a dog, but they could find commonalities that related all images of dogs together.

The third category of machine learning, reinforcement learning, is training an agent to learn specific behaviors through the process of interactions that are either returned with rewards, punishments [10], or nothing. With no prior experiences with the environment, the agent uses trial and error to find the specific sequence of actions that results in the highest rewards. Picture a dog learning how to sit for the first time. The owner uses treats

as its reward system and will provide one to the dog any time it reacts promptly to the command to sit. The first time the dog receives the treat, it is not specifically sure of its actions that led to the treat, so it starts to look back on all previous actions. During that time, the dog could have laid down, barked, jumped, or ran, so it is nearly impossible for the dog to initially connect a specific action, or sequence of actions, to be the direct cause of the treat. Through its training, the dog continues to perform many actions that result in a treat, several being received a lot sooner than others. Eventually, the dog can piece together that sitting directly after the owner's command, will lead to the maximum number of treats. By continuously reinforcing the dog's behaviors with treats, the owner effectively achieved the desired state consisting of a trained dog that sits on command.

To further expand on RL, the general idea is that an agent decides to take a specific action a based on the state of the environment he is embedded in and a specific function, named the policy function, π. This policy function is a tuple, containing of the state of the environment (*s*), the set of all actions the agent can conduct in a specific state, the probability of the actions to change the environment from its current state to a target state (*s'*), and the reward when taking a specific action in a given state. Whenever the agent has taken an action, a_t he compares the state before and after the action and adjusts his policy function π in a way, that his total reward, R_t is maximized (see Figure 1). Over the last few decades, a broad range of algorithms were formulated to implement the general idea of RL. The different algorithms can be divided into classes that differ in terms of the general procedure that is applied. Two important classes are the value-based approach and the policy-gradient approach. Within the value-based class of algorithms, the purpose is to optimize the total discounted reward by utilizing the value function. One of the more simple and most popular algorithms of this class is the Q-learning algorithm. Within the class of the policy-gradient algorithms, an "expected" reward is optimized by finding a good policy. Thereby the policy is described by a set of parameters, for instance the coefficients of a complex polynomial function [11].

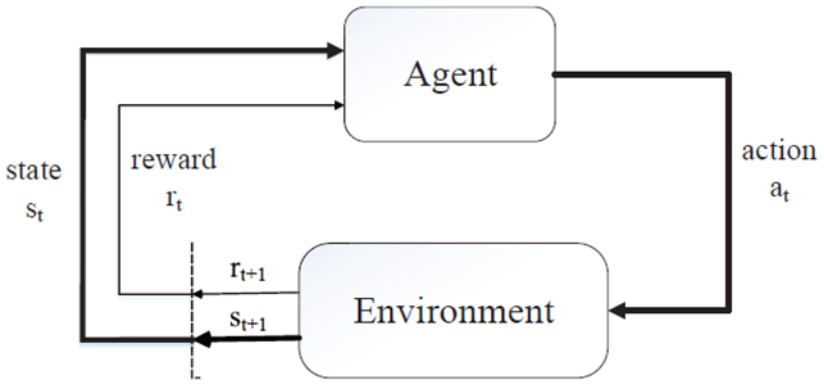

Figure 1. RL Structure. Source: [12].

A relatively new approach that has become widely used in the development of an RL application is the Proximal Policy Optimization (PPO). PPO is a reinforcement learning algorithm that uses a Policy gradient approach. It was initially developed as a refinement of the Trust Region Policy Optimization (TRPO), but PPO thus far has shown better or at least the same performance as TRPO in different applications [13]. Additionally, it has better performance figures than several other popular reinforcement learning algorithms, including the vanilla policy gradient (VPG) [14]. As with TRPO, the general idea of PPO is to improve a policy by taking large steps along the gradient of the policy function without causing performance collapse, which is the state at which the optimization algorithm does not converge anymore but diverges or becomes unstable. Unlike TRPO and many other policy-gradient algorithms, it uses a first-order instead of a second-order optimization method, thus implementing this algorithm easier [15].

While RL focuses on choosing the right action to increase a total discounted reward through trial and error, deep learning concentrates on the abstraction of higher-level features from less abstract representation observed in lower levels to include raw data. Several possible architectures to implement deep learning have been described in the literature, including Deep Neural Networks, Recurrent Neural Networks, or Convolutional Neural Networks [16]. Although there is a wide area of possible deep learning applications, including speech recognition, autonomous vehicles, or machine translation, it has lately drawn increasing attention when combined with RL to hopefully counteract ones of its

weaknesses [5],[17]. RL has been used in a wide variety of areas in recent years [18], but it still comes to its limits when dealing with high-dimensional, often continuous state spaces. In this case, neural networks' utilization to deal with high-dimensional sensory inputs offers an applicable method to mitigate RL weaknesses [11]. This combination of RL and deep learning is widely referred to as deep RL.

3. Neural Networks

Although the idea of self-learning algorithms based on neural networks was already discussed in the mid-1940s, it went in and out of fashion several times in the following decades [19]. Unlike other approaches, the idea never found its specific niche of application at that time, although they are intensively used today. There were several reasons to blame. The first is that there has been a huge improvement in the capability of the discovered and described algorithms. While some older algorithms are still powerful within their specific domain, newer algorithms tend to be more flexible and more capable of solving different types of problems [20]. This leads into the second reason. The changing culture within the IT-domain, away from proprietary software toward open-source libraries and frameworks, makes even cutting-edge software and algorithms available to a broad range of developers and users. A vast amount of documentation makes it possible that even developers and organizations with a limited specialized expertise in that technology use it to develop applicable and useful tools [20].

While these two reasons are of a more general nature and can be applied to most other IT subject areas, the following explanations stand out because of their special significance for the domain of deep RL. Due to the basic concept of deep RL, massive amounts of data are required during the training phase. With this mass of data becoming available over the past few years, this technique's application turned much more feasible [20]. Besides, even though the training data is generally produced by the RL- mechanics itself, there was still a lasting constraint that has disappeared within the last decade. Possessing this massive amount of data through necessary calculations requires enormous amounts of computational power. Through the exponential growth of technology in recent

years, the required computational power has become available, and it is much more practical to implement deep RL, even with commodity hardware [20].

To fully understand how to implement the mentioned deep RL process, the basics of neural networks must be understood. In principle, a neural network is a data structure which follows the architecture of a biological brain. In a mathematical context, a neural network is a function that can map an n-dimensional continuous input onto (generally lower dimensional) output. The entirety of this process revolves around connecting a specified number of neurons through synapses. Each neuron belongs to a specific layer, either the input layer, the output layer, or one out of one or many hidden layers. Each of these neurons are fully or partially connected with each neuron on either side of their own layer. The synapses making the connections between the layers each have their own specific weights. In addition to the weight value of the synapses, each neuron has a specific value, referred to as a bias, that is used in combination with the inputs to calculate, the degree to which the specific set of signals from the previous layer will cause the neuron to get activated [21]. This calculation is also known as the activation function. Refer to Figure 2 to see the flow between the layers.

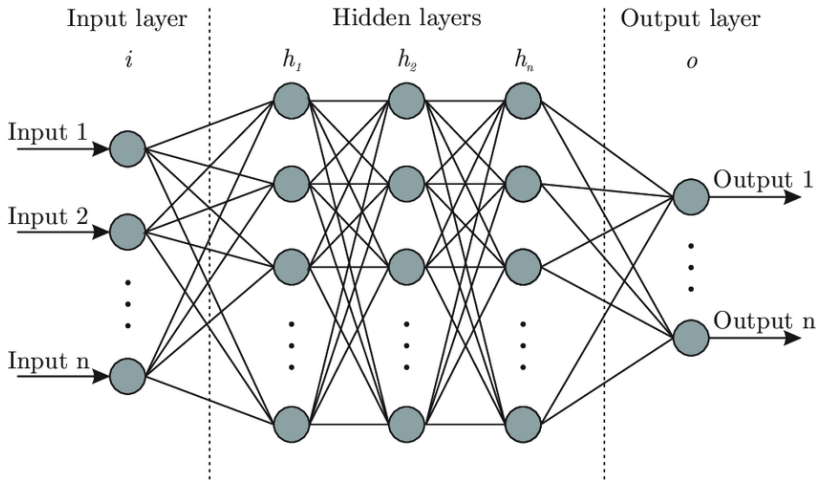

Figure 2. General Structure of a Neural Network. Source: [22].

Several different network structures for different applications and methods of training have been developed. The simplest form that follows the structure described above

includes an input, an output, and one or more hidden layers is known as a multilayer perceptron network (MLP). Learning usually works by backpropagation, meaning that after each data point of a training set is processed, the weights and biases are adjusted based on a calculated error. As soon as one data point is fully processed, the next data point of a set is sent through the neural network. Another general architecture of neural networks is recurrent neural networks. Rather than sending the data through the neural network once, recurrent neural networks process the achieved outputs back into the network as another input into the network [23].

While each of the neural network architectures mentioned above has its own field of application, they are unsuitable for classification of complex images. In their structure, traditional neural networks use one perceptron for each input that is being processed. When processing images, one pixel is equal to one input, and that is multiplied by however many pixel layers are in the image. When dealing with a basic image consisting of red, green, and blue pixel layers, the neurons required quickly start to add up and become unmanageable for the neural network. For example, an image with a dimension of 28x28 simple black-white pixels would need an input layer of 748 neurons if using an MLP. Assuming there are two hidden layers, each with 16 neurons and 10 neurons in the output layer, that sums up to 13,002 weight and bias values existent within the neural net. A picture of 200x200 pixels with 3 color channels would need 120,000 neurons on the input layer alone. A structure of this size can easily cause overfitting and result in poor training [24]. Additionally, when the images are processed through traditional neural networks, they are flattened, and all spatial variance information is lost [24]. Due to the lost spatial variance information, the neural network will not be able to identify key aspects of the image if they are in different locations or differ slightly.

4. CNN's and Multi-Agent Learning

A separate neural network architecture, Convolutional Neural Network (CNN), has recently shown success in overcoming the shortfalls that traditional neural networks have encountered when dealing with image classification. Unlike traditional neural networks, where a single vector is processed through the input layer and is then transformed through

different layers of neurons into a vector in the output layer, a CNN arranges its neurons in a three-dimensional structure (inputs or color channels, height, width), and uses convolutional layers to identify key features or patterns in each image [25]. Multiple convolutional layers can be used to together to better classify and identify key features in images. The initial convolutional layers can typically identify shapes such as edges, corners, squares, and circles in the initial layers, but as the neural network goes deeper, the later layers can detect more specific objects, such as facial features, or classify animals (see Figure 3).

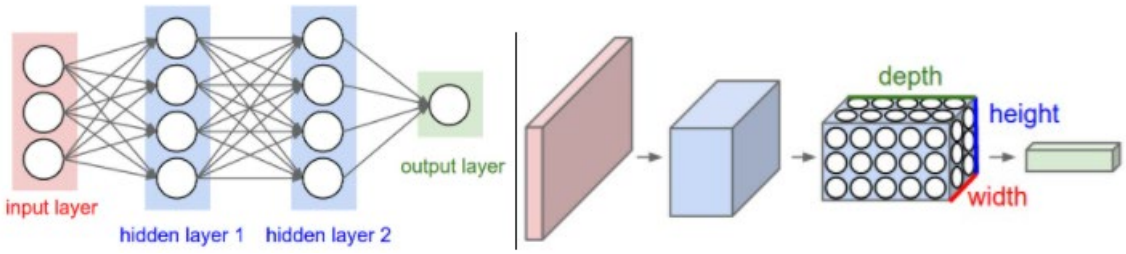

Figure 3. Comparison of MLP and CNN. Source: [25].

The basis of a CNN revolves around its use of convolutional layers as its hidden layers. In each of the convolutional layers, there is a filter, or otherwise known as a kernel, used to scan the image and identify key patterns throughout the image [25]. A filter is simply a matrix of numbers that are typically randomly initialized. The size of the filter can be whatever 2-dimensional shape designated by the user, but it is typically much smaller than the initial input. Additional parameters that the user can specify are the number of filters used, padding of zeros placed around the border of the input to control the output's spatial size, and stride, which is the rate at which the kernel will slide over the image. When receiving an input (image height, image width, number of input channels), the filter will slide over the input from the top left to the bottom right and will calculate the dot product between the filter and each set of pixels it slides over. After the full image is scanned, the output is now a 2-dimensional activation map. The convolutional layer is then usually immediately followed by a rectified linear activation function (ReLU), defined in Equation (1), introducing non-linearity into the activation map [24].

$$f(x) = \max(0, x) \tag{1}$$

Furthermore, it is common to insert a pooling layer that reduces the activation map's spatial size and the overall amount of computation in the neural networks. The most common approach to a pooling layer is using a 2x2 filter with a stride of two, that uses a max operation function over each set of values in the 2-dimensional activation map. This pooling technique can reduce the complexity of a 4x4 activation map by downsizing it to a 2x2 activation map. The last type of hidden layer in a CNN is a fully connected layer used to flatten the 3-dimensional activation map (see Figure 4). A single fully connected layer is typically placed after the last pooling layer to map the input layer to the output layer [25].

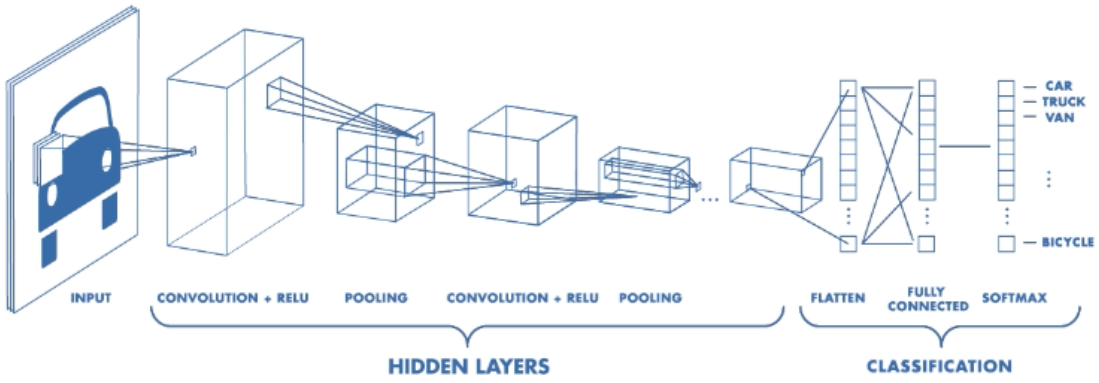

Figure 4. Fully Connected CNN, Source [26].

The use of CNNs has the ability to extend much further than basic image classification. By representing the state space for an RTS game or time-step-based simulations in an image-like manner, CNNs can be incorporated as the neural network architecture for deep RL. For instance, Egorov [27] used a CNN to properly train agents to behave in a pursuit-evasion game. The game used images like sensory information to represent the agent and environment state. The used CNN consisted of 2-layers, each consisting of 32 inputs, 3x3 filters, a stride of 1, and a ReLU. The CNN was provided four

input features: Obstacle Locations, Opponent Locations, Friendly Locations, and Self Location, which are presented in Figure 5. The information in these features was encoded as zero if empty or a non-zero integer if occupied. The evading agent behaviors were represented through a heuristic policy that moves the agent in the direction furthest from the closest pursuing agent. The pursuing agent behaviors were represented by a stochastic Q-Value policy that utilizes deep RL during its training regimen. The pursing agent's behaviors are rewarded when capturing an evading agent through occupying the same grid square. When multiple pursuing agents are operating in the same environment, they must cooperate to capture the evading agents in the shortest amount of time. During the training phase, one agent was trained at a time, while all the other agent's policies were kept fixed. After a set number of iterations, the trained policy got distributed to all of the other pursuing. This process allowed the agents to continuously build on and improve their performance over time.

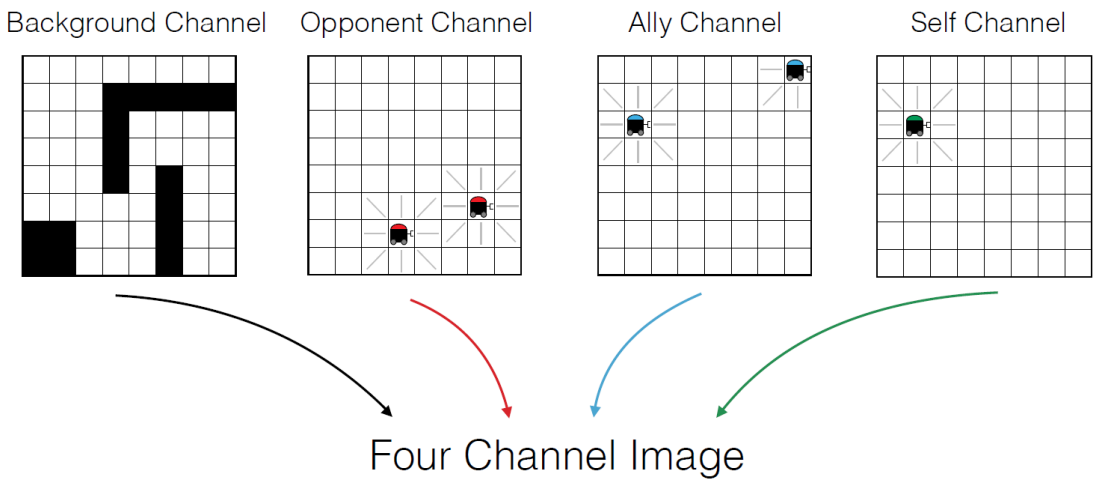

Figure 5. Example of State Representation Using Four Input Channels. Source: [27].

For evaluation, Egorov [27] demonstrated the agent's ability to generalize training for multiple scenarios. A pursuing agent was trained on two different environments simultaneously, each having a different obstacle configuration, as shown in Figure 6. Once training was complete, the policy was evaluated in a new environment that combined the

two obstacles. The agent successfully generalizes the training on the two separate instances and achieves similar results to an agent trained solely on the evaluation environment. Additionally, the agent's ability to generalize training with multiple pursuing and evading agents in the same scenario was evaluated. Three policies were trained for 1 vs. 1, 2 vs. 2, and 3 vs. 3 scenarios. Each policy was evaluated on all three scenarios. For the 1 vs. 1 scenario, there was no need for cooperation between the agents, so all three policies could achieve optimal performance. When increasing the complexity in the 2 vs. 2 and 3 vs. 3 scenarios, both policies had the best performance on their respective scenarios, but both still outperformed the 1 vs 1 policy [27].

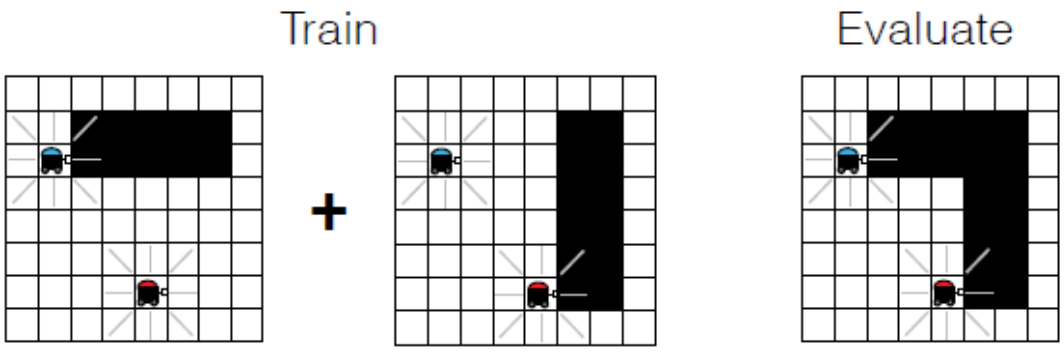

Figure 6. Training and Evaluation Environment in a Pursuit-Evasion Game. Source: [27].

In general, an agent trained by reinforcement-learning techniques is limited by the complexity of the used environment itself. This deficiency can be addressed with a multi-agent-learning approach, where the agents are learning off of each other's behaviors rather than pre-written scripts. Multi-agent learning approaches show two valuable properties. First, it is possible to train agents that can show highly complex behaviors even though the environment in which they operate is based on a limited number of simple rules. An example of this is DeepMind's AlphaGo [28], where Go's environment is limited heavily through the rules of the game. The complexity needed to train a highly performant agent is introduced into that environment using a second agent as the opponent. The other valuable property is that through multi-agent RL, the agent will be trained in a variable environment

according to its difficulty. As its performance increases, the agent will be forces to solve more complex as the skill level of its opponent will also increase. Through continuously matching opponents with comparable skills levels in multi-agent RL [29], the agents will continue to exploit its opponent's vulnerabilities and successfully perform more complex task.

In many recent papers, multi-agent reinforcement learning is separated into two phases, the *exploration curriculum* and a *competitive phase* [29]. During the exploration curriculum, the agents must be initially trained from a state with no prior interactions with the environment. A common approach that was used by Bansal et al. [29]. involves the agent learning all of its information about the environment by simply interacting with its surroundings. There is nothing injected into the environment to guide the agent's decisions other than the reward system itself. A more intricate approach that has shown success in quickly training an agent in recent years uses supervised learning from old replays to train the agent initially [5]. Here the agent will mimic the behaviors shown to have success from prior plays, but once it learned every technique from that replay, the agent will then explore unidentified methods that will hopefully produce better results. With that approach, comes two disadvantages. In the case of commercial games, this solution is feasible as the developers likely have access to the replay data from thousands of played games. For other instances, including military simulations, a library with this amount of data is often unavailable. The other disadvantage is that human players might tend to apply a not optimal strategy to a problem. The agents trained with that replay data might be biased toward a solution that they would not choose if trained without such negatively biased data.

During the competitive phase, the already trained agents will be improved by playing against other trained agents. In this phase, matchmaking has a huge impact on the trained agent's ability to deal with complex situations. An important aspect of this phase is to ensure that an already trained good AI does not "forget" bad gameplays from weaker opponents. A possible strategy to prevent that is to let good agents play against agents of weaker skill frequently [5].

B. STATE OF RL RESEARCH IN RTS GAMES AND MILITARY SIMULATIONS

Since the emergence of AI in the 1950s, people have consistently tried to find ways to measure its performance against human beings. Games are a common tool that researchers have resorted to as their platform since it allows them to measure and compare performance in real-time while also often resulting in a distinct winner. Therefore, the following chapter will first give an overview of latest successful utilization of AI, particularly deep RL, within the game industry. In a second step, the current research in regard to deep RL within military constructive simulations is described.

1. AI in Games

As a game that is accepted worldwide for requiring a large amount of intelligence and strategic ability, chess has gained a lot of attention from researchers seeking to develop an AI agent that can outperform top-level players. In 1997, IBM's computer, DeepBlue [30], was the first to accomplish this feat by defeating the world chess champion at the time, Garry Kasparov, with a final score of 3.5 to 2.5 [30]. In 2017, researchers directed their focus toward Go, a game much larger in complexity than chess and often deemed too challenging for a computer to master. DeepMind Technologies' AlphaGO, which was trained using previous human and computer play combined with deep learning techniques, defeated one of the world's best Go players, Lee Sedol [28].

Despite researchers achieving success developing AI agents capable of defeating the best players in games where both players are visible to all information inside a state at any given time [31], games that withhold private information, such as poker, had yet to be tackled. Developing an AI agent in environments that contain private information only available to that individual player increases the complexity drastically. Simple solutions that try to search for an optimal sequence of actions are no longer applicable since certain information must not be revealed to their opponents. Using an approach that reassesses its strategy and weighs the probability of success after each decision, Brown and Sandholm [31] developed an AI agent, Libratus, that triumphantly defeated four professional players in no-limit Texas hold'em.

With AlphaStar, OpenAI has recently shown the power of an approach based on the connection of ordinal neural networks and a long-short-term memory network (LSTM) in an environment, far more complex than environments solved with comparable approaches so far. An action space including the control of hundreds of units and their possible actions and a state-space build-up from different feature planes result in 10^{26} possible choices at each decision point. Also, a game structure where the final success is dependent on an overall strategy that is played over a thousand steps made the development of this StarCraft II AI way more complex than solutions implemented before for applications like Go or the ATARI games. Nonetheless, OpenAI achieved to train an agent who could beat 99.8% of the players ranked on the game-platform battle.net [5].

One reason for the success of OpenAI in StarCraft II was their approach of using multi-agent training. Having available the datasets of 971,000 replays of matches between human players, they trained their agents in a supervised learning approach based on the gameplay of humans that belonged to the top 22% percent of all players. They then reduced the exploration problem that appears when training agents from scratch with a random exploration schedule. In a second step, they trained their agents against each other. To prevent learning cycles, they used a complex matchmaking process in which their agents were divided into different subgroups. Based on that subgroups, they ensured that even high-performing agents regularly got weaker opponents, thus not "forgetting" how to react when less efficient strategies are played [5].

Defense of the Ancients 2 (DotA2) is another multiplayer real-time strategy game (RTS) for which OpenAI successfully developed an AI algorithm based on neural networks. Brought onto the market in 2013, it averaged between 500,000 and 1,000,000 players between 2013 and 2019. With an active community of professional players and over $35 million in prize money during the 2019 international championship, it is one of the major massive online player games currently on the market [32].

As an RTS game, it has specific properties that make RL for that game significantly different than more "traditional" style games like chess, Go, or even arcade games. One of the main differences is that RTS games have a much longer time horizon than other games. While a usual chess game lasts approximately 80 moves, an average DotA2 game runs over

45 minutes with 30 frames per second. Assuming the AI will decide on the next action every fourth frame, that still sums up to over 20,000 steps whenever a decision has to be made. Moreover, unlike in chess, Go, or arcade games, the game's state is only partially observable, so an agent must successfully make assumptions about the enemy's behavior based on incomplete data. The last difference revolves around the number of dimensions present in the game's state. While chess, Go, and comparable games are quite limited according to input-variables and possible course-of-actions, in RTS games, the AI has to deal with a vast amount of input parameters, including different unit types, map data, building options, etc. Furthermore, after processing this amount of input data, the AI finally has to choose one action out of a vast range of different actions. In DotA2, these possible actions range between 8,000-80,000 in each step [32].

The developed AI successfully competed against professional players in the 2019 international DotA2 championship. The AI's foundation is, similar to AlphaStar, an LSTM with a single layer comprising of 4096 units. The multi-dimensional inputs from a current game state are put into a single vector that serves as input for the LSTM. Within this LSTM, the one-dimensional input vector is processed, leading to an output based on which the next action is chosen in combination with a policy function. Unlike in AlphaStar, the AI does not control all units with one neural network. Similar to the nature of the game where different players each control their own hero, each hero is controlled by a replica of the described LSTM. Variations in the chosen action result from a different observation space the various players face due to the visibility of information and the fog of war [32].

2. Neural Networks in Military Applications

Based on the successful developments related to games, advanced deep RL approaches were applied for military research. One application that lately drew the attention of the publicity was an AI developed within a Defense Advanced Research Project Agency research project. The developed AI algorithm, called AlphaDogfight, was trained to successfully control a fighter plane in a dogfight against a real human pilot, displaying tactics that resulted in the AI winning all five runs [33].

Within the military constructive simulation domain, Moy and Shekh [34] utilized an AlphaZero-algorithm as the agent to play the Hexagon-wargame Coral Sea, a wargame used by the Australian Defense Forces. It is a turn-based game that is played on a hexagonal-grid board with two players (see Figure 7). To win the game, a player has to achieve defined objectives within a specific number of rounds. A turn consists of three phases. In the movement phase, players move their units from one hexagon to another. It is possible to have more than one unit occupy the same hexagon. In the acquisition phase, players define which enemy units their units shall target and fire upon. In the fire phase, the firing order from the acquisition phase are executed. After each turn is finished, the *Initiative Card* is turned over to the other player. The player holding the *Initiative Card* goes first during the movement and acquisition phase, but second during the firing phase. In their research, Moy and Shekh used a simple scenario with one red and one blue unit. The red unit was initially posted on the top left corner of the board, the blue unit in the lower right (see Figure 8). The red unit had to defend its position; the blue unit had to reach the red starting position. When both players conduct no action, red would win this game by default. However, due to players' changing order of precedence between the acquisition and firing phase, the blue player would always win when showing optimal behavior.

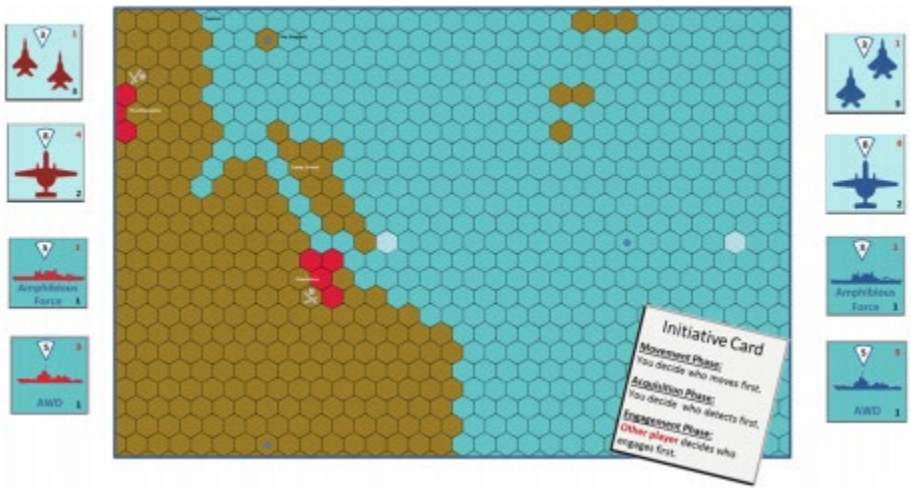

Figure 7. Coral Sea Board Game. Source: [34].

AlphaZero was originally developed by DeepMind and trained to play games like Go or chess and consists of a neural network combined with a Monte Carlo Tree Search (MCTS) algorithm. The algorithm starts with an "empty" neural network that is only provided the basic rules of the game. Using this approach for conducting training from scratch in a given scenario, many computational resources are often required. This requirement stems from the wide range of exploration needed to calculate the possible moves between the blue player's initial position and destination. Moy and Shekh implemented three predefined behaviors to solve this problem: *SafeGoalMove*, *GoalMove*, and *RandomLegal*. SafeGoalMove is used to move into the next closest hexagon to the objective and only moves into the range of enemy fire if the *Initiative Card* is held. GoalMove is used to move to the next closest hexagon to the objective without when the *Initiative Card* is not held. A *RandomLegal* behavior is any move allowed by the game rules.

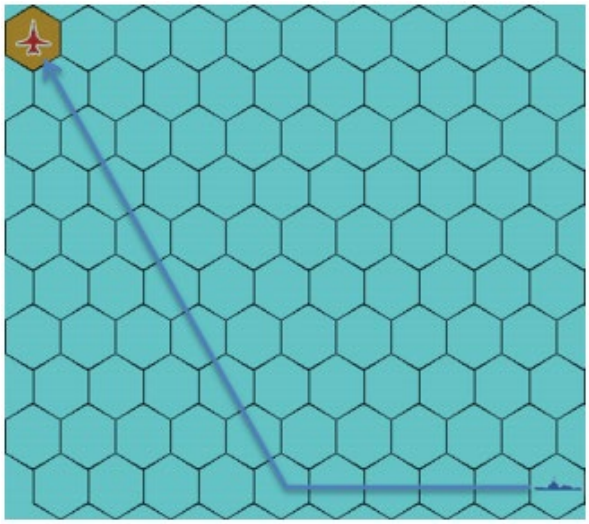

Figure 8. Used Coral Sea Scenario. Source: [34].

During the training phase, an agent now takes one of the above strategies using a probability of P_h. If 1-P_h is selected, the traditional MCTS solution from the Alpha Zero-approach is used. Using this strategy, it was possible to reduce the training time and computational resources down from approximately 24 hours to 15 minutes.

Additional research using neural networks in constructive military simulations was completed by Sun et al., where they used Prior Knowledge-Deep Q Network (PK-DQN) within a constructive hex-based military simulation [7]. In this simulation, two parties, red and blue, each command units to move across a battlefield and fire once their opponent is within range. Each player's goal is to win a specific hex, or waypoint, marked by a flag, as shown in Figure 9.

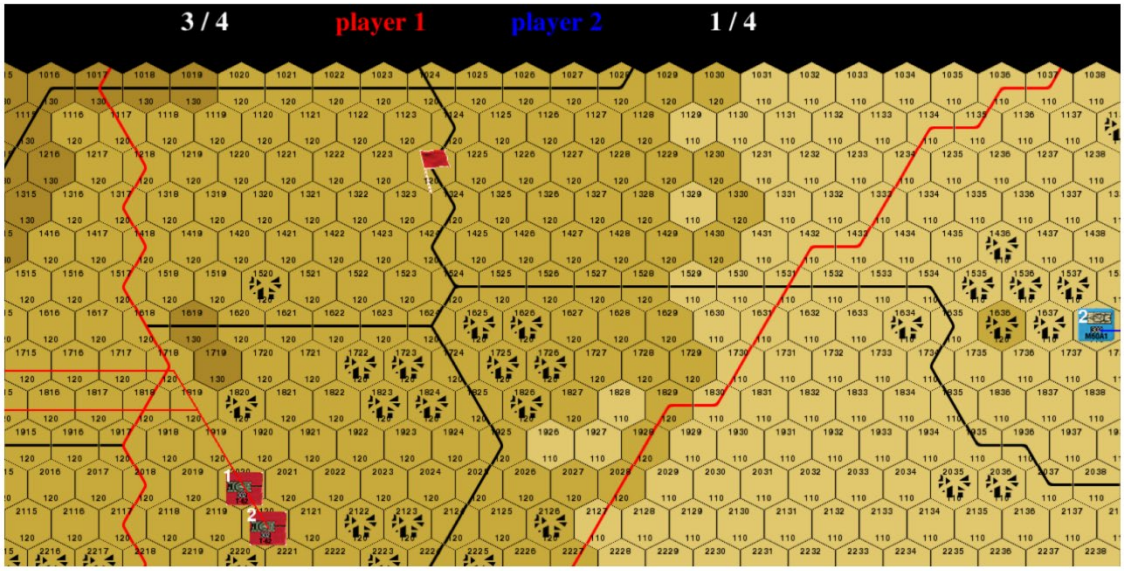

Figure 9. Structure of an Underlying Hex-Based Simulation. Source: [7].

Instead of using the PPO RL algorithm to train the agent, Sun et al. utilized a modified approach. Typically, Q-Learning takes all possible actions for a unit on the battlefield at each specific state and places them into a table with the expected reward for that action-state combination. Unfortunately, this approach in complex environments causes the table to increase drastically with each additional game state or possible action added to the scenario. To deal with that shortcoming of a traditional Q-Learning approach in complex scenarios, DQL was developed. Here, the Q-learning table is replaced by a neural network that is utilized to determine an agent's next action given the current state [35].

DQL has recently proved to be a successful approach for a huge range of problems. However, for their simulation, Sun et al. identified a high level of randomness between the games and an overall slow speed of convergence. To address this problem, they introduce "prior knowledge" into the learning process. "Prior knowledge" is defined as a function, mapping a set of specific characteristic states $P=\{p_1..p_n\}$ onto an optimal action $a^* \in \{a_1..a_n\}$. Whenever a given state S is an element of P, the mapped optimal action a*, is used as final action instead of the original estimated "deep learning" solution. For a set of generic states, the action an agent takes is not determined by a trained neural network anymore, but by a set of pre-defined rules (Figure 10). This causes the number of explorative moves, especially at the beginning of each game, to be reduced drastically.

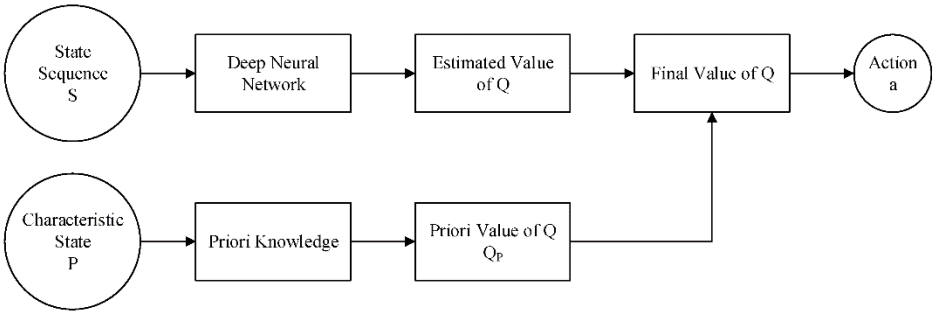

Figure 10. General Architecture to Include a-Priori Knowledge into Learning Process. Source: [7].

For evaluation, Sun et al. created a scenario in which an agent, trained by PK-DQN, plays against a rule-based opponent. The results are compared with results produced by utilizing an unmodified DQN for training. With the modified PK-DQN, stable results are achieved much faster than with an unmodified DQN. While the traditional DQN approach required 33 hours, a PK-DQN approach reached similar results after only 23 hours.

Sun et al. used a hex-based grid as the foundation for the simulation where a unit is either located in a hex-field or not, which allowed for a basic state representation for the units. Another possible approach was conducted by Boron [6]. In his simulation, entities move over a plain battlefield consisting of featureless terrain. For the battlefield representation, a rectangular grid where each grid section hosts a sensor in its center is

used. By calculating the distance of a unit to each of the 4 closest sensors and normalizing this distance over the sum of the four distances, he determines a rational value indicating if and how much of a unit is positioned in a grid cell (see Figure 11). Boron's question in his research focuses on whether an AI agent utilizing an MLP framework can be trained to apply the economy of mass and economy of force within simple military scenarios. In his results, Boron trained AI agents to show stable optimal tactical behavior within simple 2 versus 1, 2 versus 2, and 3 versus 2 scenarios. Using different optimization algorithms, he found that TRPO might outperform PPO and VPG when applying deterministic combat models. Furthermore, trained AI agents learn to apply the tactical principle of mass or economy of force depending on the discount factor used during the training.

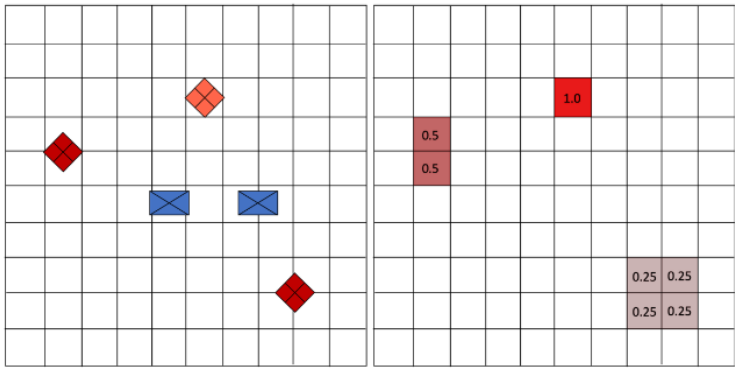

Figure 11. Actual Position and State Representation of Units in a Rectangular Based Simulation. Source: [6].

THIS PAGE INTENTIONALLY LEFT BLANK

III. FRAMEWORK

Within the last chapter, basic concepts such as machine learning, neural networks, and multi-agent-learning were explained. Moreover, the current state of research within the field of constructive simulations was presented. In this chapter, the used simulation software, called Atlatl, will be explained. Atlatl is an AI training environment developed within the MOVES Institute at the Naval Postgraduate School (NPS). In the first part of the chapter, the basic architecture will be explained and the reason for using different external software packages is given. In the second part, the capabilities and general rules of Atlatl will be laid down.

A. ARCHITECTURE

Atlatl is implemented in a client-server architecture that uses JSON for communication between the participating entities. The server and AI clients are both implemented in Python, as it offers a large variety of open-source projects, and it is popular throughout the deep RL research community. JavaScript is used to create the user interface needed for human client interactions. To illustrate the hexagonal grid and the unit icons in a graphical representation, Scalable Vector Graphics (SVG) [36] is incorporated. SVG is a specification developed by the World Wide Web Consortium that allows the two-dimensional vector display to be easily represented in an HTML browser. SVG stores the exact coordinates for each part of each object represented, enabling the graphics to be easily scaled. The ease of scaling vectors allows SVG to have clearer graphics over a bitmap image. The architecture for a match including a human player versus an AI player is shown in Figure 12.

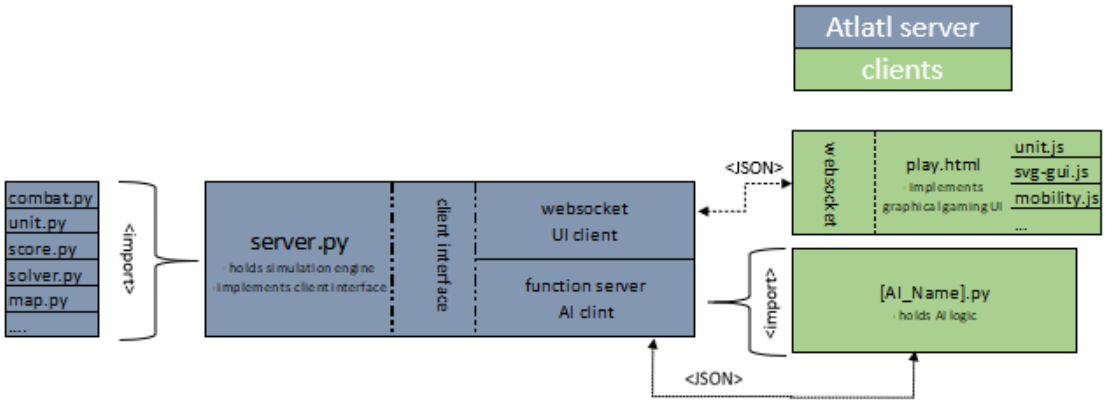

Figure 12. Architecture Human versus AI Match.

Stable Baselines3 was selected as the RL resource for this research. The Stable Baselines3 library is an improved version of its original project, Stable Baselines, a fork of Open AI's Baselines library that creates a common interface for many RL algorithms and includes simple and understandable documentation [37]. Although Stables Basslines3 can be used on all operating systems, it is advised for Windows users to utilize Anaconda, an open source Python distributor that allows for an easier installation of packages and required libraries [38]. Python 3.6+ and PyTorch 1.4+ installations are required before using Stable Baselines3. The Stable Baselines3 library currently implements twelve different RL algorithms, including multiple on-policy and off-policy algorithms, that can easily be bound into self-developed Python code [38]. As with Open AI's Baseline, the Stable Baselines3 algorithms can only be applied in environments that are compliant to the OpenAI's Gym library. As such, a Gym-compliant interface is implemented in Atlatl that serves as bridge between the original Atlatl environment and the algorithms implemented in Stable Baselines3. The architecture for a training setup utilizing Stable Baselines3 is shown in Figure 13.

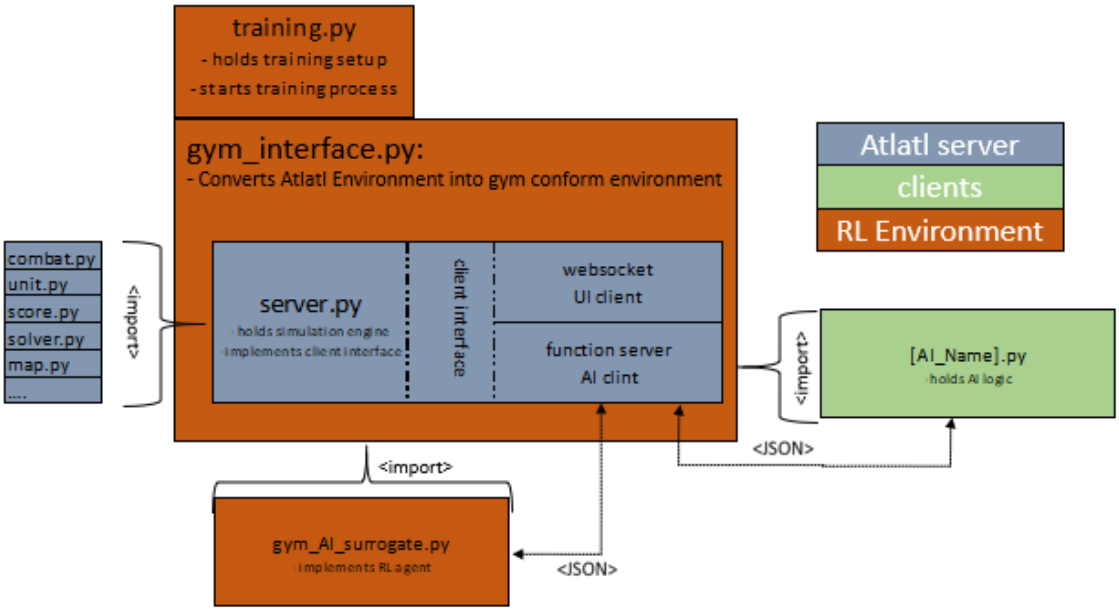

Figure 13. Architecture Utilizing Stable Baselines3 for Training.

The steps involved during a training run with the shown architecture replicate the same agent-environment loop discussed previously during the explanation of RL, in which an agent chooses an action in an environment and is returned with an observation and award [39]. When using Gym, the agent-environment loop is implemented using four values that consist of the observation the agent makes, the reward it receives, a Boolean representing the status of the environment, and diagnostic information that can be used to improve the environment [39].

B. TRAINING ENVIRONMENT

The developed training environment allows the representation of different unit types and various terrains in a two-player turn-based wargame. The tiles of the map are in hexagonal shape, and the map size is not restricted. Combat outcomes for each scenario are determined by a deterministic Lanchester model. In each turn, a player can order a unit either to move or to shoot according to the range restrictions applied to the units' type and location. Each player takes successive turns, where the acting player may take one action for each of its units. The order in which each unit acts is based on a pre-determined fixed rotation, which is pre-determined when creating the order of battle for the scenario. Both

players aim to maximize their score, where the blue player maximizes a positive score value, and the red player maximizes a negative score value. A game always begins with a setup phase and ends if either one player has all units destroyed or the maximum number of turns is reached.

Currently, there are four-unit types allowed in the player's force structure. The unit types include infantry, mechanized infantry, armor, and artillery. These units are represented as entities at the regimental level and will abide by NATO Joint Military Symbology when graphically displayed in the simulation, as shown in Figure 14.

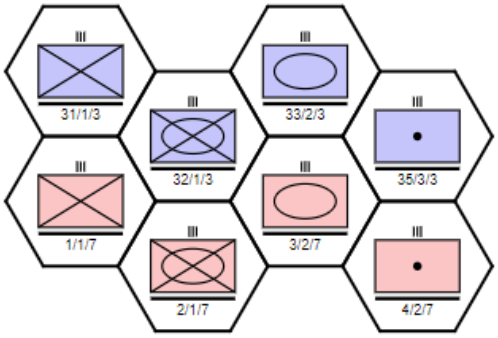

Figure 14. Unit Types (Infantry, Mechanized Infantry, Armor, Artillery).

When creating a simulation scenario, five separate terrain types can be selected: clear, water, marsh, rough, and urban. Figure 15 shows the graphical representation of the different terrain types in the simulation.

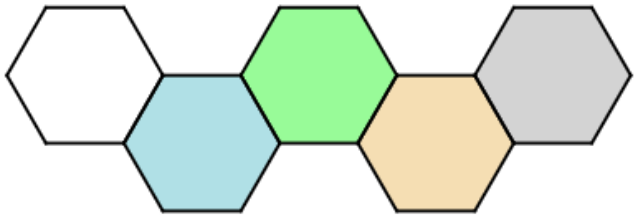

Figure 15. Terrain Types (Clear, Water, Marsh, Rough, Urban).

The clear terrain type can be thought of as a flat plain that allows freedom of movement through its space for all unit types. Marsh terrain represents the wetlands typically found at the edge of rivers and lakes, full of high grass and areas that restrict movement. The rate and distance at which a unit can move through different terrain types are determined by a mobility value. This value indicates the percentage of a turn required for a unit to navigate through a specific terrain type. This value is also dependent on both the terrain and unit type, as shown in Table 1. For example, an infantry unit moving into clear terrain has a mobility factor of 100, indicating that the unit cannot move anywhere during that move since 100 percent of its time in the current turn is needed to conduct the moving order. Armor units have a mobility factor of 50 when moving into clear terrain though so they can conduct a second move into another clear terrain if desired. Artillery units are not able to move into marsh terrain, indicated by 'NA' in the table. All unit types cannot move into water.

Table 1. Mobility Adjustment Based on Unit Type and Terrain.

Mover	Terrain Entered			
	Clear	Rough	Marsh	Urban
Infantry	100	100	100	100
MechInf	50	100	100	100
Armor	50	100	100	100
Artillery	50	100	NA	100

For each hexagon, it is determined for which side it can be utilized as setup hexagon during the setup phase. A blue dot indicates a possible setup hexagon for the blue player, a red dot a possible setup hexagon for red player. Which hexagons are finally chosen by the player as starting hexagon for his units is determined by the used AI respectively by the human player (see Figure 16).

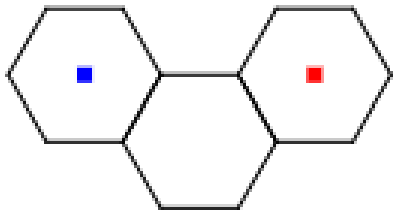

Figure 16. Setup Hexagons (Blue Player, No Setup Hexagon, Red Player).

When entering combat with the opposing force, all unit types have the ability to fire into neighboring hexagons. However, as an indirect fire weapon system, artillery units have a fire range of two hexagons. The rate of attrition (*ATTR*) for an attacked unit in combat is modeled by the attacker's strength ($ATCK_{Str}$), the attacker's firepower (*FP*) based on attacker and target unit type ($ATCK_{Type}$/ TAR_{Type}), and the defender's bonus (*DB*) based on target unit type (TAR_{Type}) and the terrain type the target unit is located in ($TAR_{Terrain}$). The value gotten based on that numbers is finally adjusted by a scaling factor (SF) (see Equation (2)). The scaling factor was set to 0.5 during the whole thesis research.

$$ATTR = ATCK_{Str} \bullet FP\left[Atck_{Type}, Tar_{Type} \right] \bullet DB\left[Tar_{Type}, Tar_{Ter} \right] \bullet SF \qquad (2)$$

Each unit type has a separate strength scale determined based on their target's unit types. This simulation provides an advantage to armor by giving it the most protection and a disadvantage to artillery with it having the least protection. Table 2 describes the entire scaling relationships between each unit type and shows how the firing power is determined based on attacker's and defender's unit types. While in combat, once each unit reaches a strength level below 50 percent, the unit is deemed combat ineffective and is removed from the simulation.

Table 2. Firing Power (FP) Based on Shooter and Target Type.

Shooter \ Target	Infantry	Mechinf	Armor	Artillery
Infantry	1	1	0.5	1.5
MechInf	1	1	1.0	1.5
Armor	0.5	0.75	1.0	1.0
Artillery	1	0.75	0.5	1.5

When a non-infantry unit does occupy marsh terrain in the simulation, it receives a defensive disadvantage as it would be more exposed and have limited areas to navigate. This is modelled by a defender's bonus (DB) of 2 (see Table 3). When an infantry unit occupies a rough or urban terrain hexagon however, it is considered that their foot mobile troops will have more objects to use as cover. Thus, they receive 50 percent less damage in the simulation.

Table 3. Defender Bonus (DB) Based on Unit Type and Terrain Type.

Target \ Target's Terrain	Clear	Rough	Marsh	Urban
Infantry	1	0.5	1	0.5
MechInf	1	1	2	1
Armor	1	1	2	1
Artillery	1	1	2	1

The score of a game is calculated based on the losses each side suffered. For each point of attrition, the blue player inflicts onto red units, +1 is added to the game score. For each attrition inflicted onto the blue units, -2 is added to the game score. Control over city hexes generate an additional change of the score value, positive if blue owns a city and negative if red owns a city.

C. STATE REPRESENTATION

As mentioned in Chapter III.A, the utilization of Stable Baselines3 presupposes a OpenAI Gym conform RL environment. With that, it is necessary to transform the native Atlatl state and action space into a representation than can be used for RL. To deal with some disadvantages in regard to the coordination system in hexagonal based simulations compared to a rectangular grid system, Atlatl uses a double coordinate system [40]. In a usual offset hexagonal coordinate system, the hexagon located at the top left of the grid would be labeled as (0,0), the hexagon adjacent to the right of it as (1,0), and the hexagon in the third column as (2,0). In the double coordinate system, coordinates in the y-axis direction will be incremented by 2. For instance, the hexagon perpendicular below the (0,0)-hexagon is the (0,2)-hexagon, and the (0,4)-hexagon follows that. With that pattern, directions of movement can be directly translated into a new hexagon coordinate. A movement to the top-left always comes with a decrease of 1 on both axes. In a coordinate system without double coordinates, the change on the y-axis would be dependent on whether the x-value is odd or even (see Figure 17).

Figure 17. Hexagonal Board without/with Double Coordinates. Source: [40].

The state space provides by the Atlatl framework based on the double coordinate system will be transformed into a Gym conform observation space by the implemented Gym interface. By default, the neural network is provided three inputs to represent the state space in the simulation. These inputs include the location of the moving units, the locations and strengths of all blue force units, and the locations and strengths of all red units. A fourth input was added during the research to include a terrain's locations and identity. The different inputs are represented as arrays. For its representation, a 1.0 or strength value is used to represent if a hexagonal tile is occupied, and a 0.0 is used to represent the space as empty. Due to the formatting when using a 2-dimesional array, the x-axis and y-axis are flipped for the inputs. To ensure a fixed convolutional kernel can see the same set of neighbors regardless of where it is centered when using 2-dimensional arrays, an additional row is added, and objects located in odd columns are represented one row lower than objects in even columns (see Figure 18).

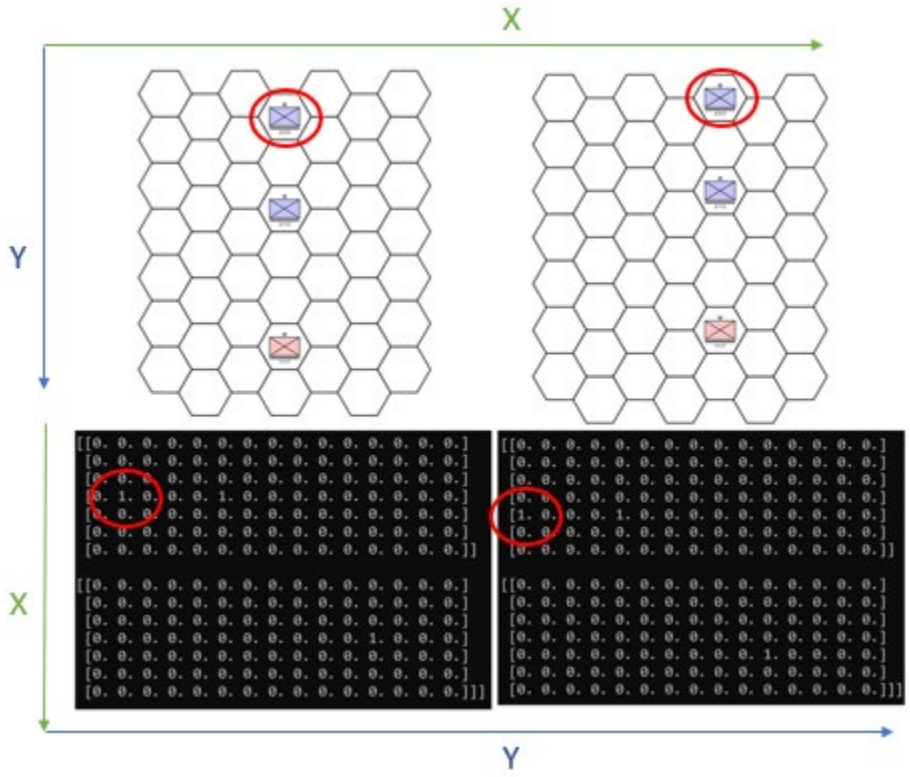

Figure 18. Stretching of the Y-Axis.

Figure 19 demonstrates the full observation space given to the neural network for the shown situation in the simulation. The top array holds the information which unit is the acting unit. The unit in the top left on hexagon (2,2) is the first unit in the turn. This is represented by a 1.0 in the third row, third column of the array. The second input is the locations and strengths of all blue units. Since both units still have full strength, there is a 1.0 in the corresponding positions of the second array. As for the blue units, the third array holds the information about location and strength of all red units. After each unit movement, the features are updated to correctly reflect.

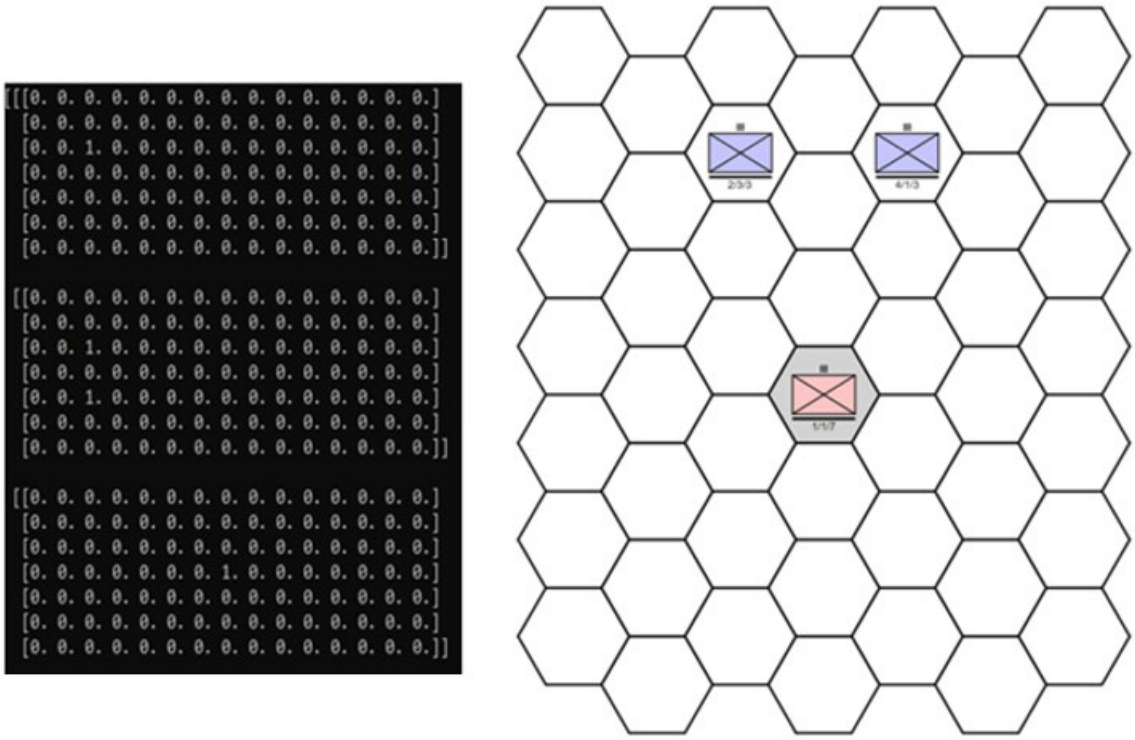

Figure 19. Complete State Space Representation with Three Inputs (Mover, Blue Units, Red Units).

Based on the given inputs, the neural network decides on an action in accordance with the rules of the wargame. The used action space is discrete numbers ranging from 0 to 18, representing the 19 possibilities a unit can choose as its next move or fire target (see Figure 20).

If the neural network selects an action outside the boundaries for the acting unit, that action is suppressed, and the unit will receive a hold older to remain in the same hexagon for that move.

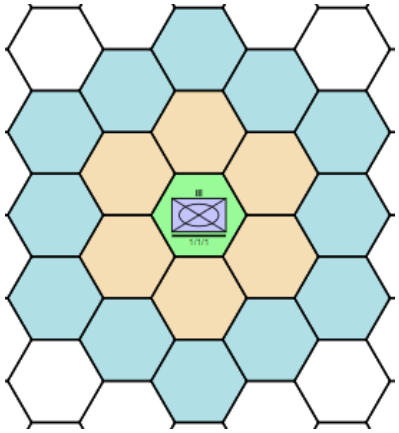

Figure 20. Action Space (Green: No Action, Brown: Move/ Fire with Distance 1, Blue: Move/ with Distance 2).

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SCENARIOS AND RESULTS

In the last chapters, an overview of theoretical concepts used in this thesis was provided. Moreover, the state of current research within the game industry as well as in the domain of military constructive simulations was laid down. After that, the used simulation framework, called Atlatl, was described. In this chapter, the scenarios investigated within this thesis are described. The observed AI agent's behavior is explained, and possible conclusions are formulated. In total, six scenarios are investigated. In a first step, the results gained from the scenarios used by Boron [6], 2-versus-1, 2-versus-2 and, 3-versus-2, are shown. While the starting positions of the blue and red player and the red player's behavior are fixed in Boron's scenarios, the scenarios in this thesis will bring in variability during the blue and red players' setup phase and in the red behavior. In a second step, the results gained from more complex scenarios are shown. These scenarios differ from the first three scenarios in complexity due to different unit types, terrain types, or multi-agent training approaches.

For all scenarios except the multi-agent training, the blue player was the only player trained by the neural network. The number of turns played in the game was by default set to 20. As for scoring values the default settings of -2 and 1 are used (see Chapter III.B.). To account for the red force has an embedded advantage within Atlatl's structure, changing the score multiplier to -2 points instead of just -1 point for each strength point caused by the blue force helps simulate the existent defensive superiority. For the red player, the following hard-coded AIs are used:

- "passive": A passive red AI does not move at all. Furthermore, it does not react when blue units enter adjacent hexagons.

- "shootback": A shootback red AI does not move at all. However, whenever a blue unit is in an adjacent hexagon, the red unit opens fire. If there is more than one blue unit in the adjacent fields, the target unit is selected randomly.

- "withdraw": A withdraw red AI only moves when two or more blue units are in adjacent hexagons. If no blue unit is adjacent, the red unit does not move at all. If there is one blue unit adjacent to the red unit, the red unit opens fire.

- "random move": A random move red AI moves with a chance of 40% into a random direction. However, as soon as blue units are within adjacent hexes, it opens fire as if it were a shootback AI.

As an optimization algorithm, PPO is known for its forgiveness when conducting hyperparameter initialization [41]. As the focus of this thesis is not to maximize the use of the neural network, but to explore the RL capabilities in constructive simulations, PPO seemed to be an excellent fit. The learning parameters were kept constant during all scenarios with a learning rate of 0.0003 and a clip range of 0.2. For the underlying neural network architecture, a 3-layer CNN is used (see Table 4). For the first layer, a kernel size of 3 by 5 was chosen to adjust for the used double coordinates, where the y-axis is stretched in the underlying hexagon board structure.

Table 4. Used CNN-Architecture.

Layer	Input Channels	Output channels	Kernel size	Stride	Padding
1	3	32	(3,5)	(1,2)	2
2	32	64	(3,3)	(1,1)	1
3	64	64	(3,3)	(1,1)	1

The reward system used to reinforce the behavior of the agent during training incorporates the Atlatl score value achieved by the player at the end of each turn. However, this value is modified. A reward is only given to the agent if the score value during that turn is greater than 0. All rewards greater than 0 are then discounted based on the number units still alive compared to the number of units at the start of the scenario (see Equation (3)). The same reward structure is used for all scenarios.

$$\text{if } score\ value > 0,\ Reward = score\ value * \frac{number\ of\ blue\ units\ alive}{total\ number\ of\ blue\ units} \tag{3}$$

A. TWO-VERSUS-ONE

1. Description

The training scenario entails a two-versus-one configuration comprising two blue units attacking a single stationary red unit on a 7x7 terrain. The scenario is used in four different configurations. In the first configuration ("fixed"), all units are set up in a fixed absolute position on the battlefield. The shootback AI is used for the red force behavior. For the second configuration ("fixed formation"), the first configuration is modified so that the relative position of the units during the setup phase is kept constant. Still, the whole formation is shifted randomly over the battlefield (see Figure 21).

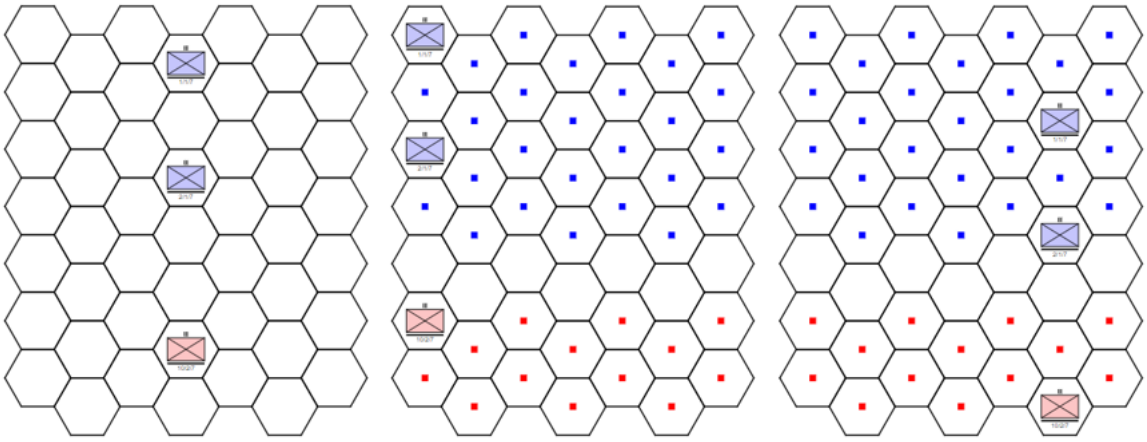

Figure 21. Two-versus-One Configuration with Fixed Staring Positions (left) and Changing Position of the Whole Units' Formation (middle/ right).

In the third configuration ("random"), the units are randomly positioned on the allowed setup fields and the red force's behavior are still described by the shootback AI (see Figure 22). For the fourth configuration ("withdraw"), the units set up positions are consistent with the second configuration. However, instead of a shootback AI, a withdraw AI is used for the red forces.

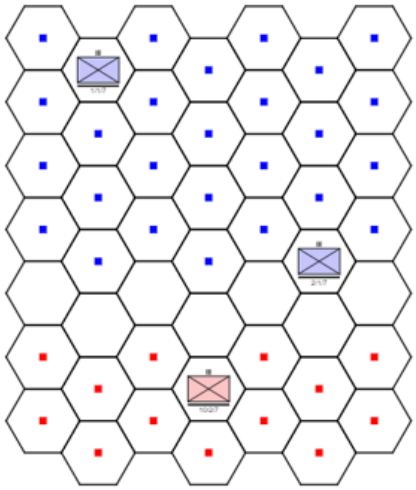

Figure 22. Example of Possible Setups in the Two-versus-One Configuration 3.

Due to the simple configuration of the scenarios, the optimal score was manually calculated. In the first three configurations, the maximum score for the blue force is achieved when it manages to move both units next to the red unit simultaneously. In this case, the red force will cause 50 damage to the blue force in an initial attack, leading to a scoring value of -100 points. In the next turn, both blue units attack the red unit causing 75 damage points. However, since the red unit is below the strength threshold of 50 percent, the unit is deemed combat ineffective and is removed from the simulation. Resulting from the attack, the blue force scores 100 points, leading to a final game score of 0 points and a remaining strength value of 150. The reward used for the training process in this scenario is slightly different from the default reward previously described. Instead of discounting the reward based on the number of remaining blue units, the discount factor is calculated as the quotient of the remaining blue strength divided by blue strength at the start of the game. The optimal reward in these three configurations is 75 (see Table 5).

Table 5. Two-versus-One Optimal Behavior Performance Values

Configuration	Optimal Blue Strength	Optimal Total Discounted Reward	Optimal Score
fixed	150	75	0
fixed formation	150	75	0
random	150	75	0
withdraw	200	100	100

In the fourth configuration ("withdraw"), the values differ. This is because an optimal behaving blue force will force the red unit to retreat by attacking simultaneously with both units. As result, two situations can occur. In the first situation, the red unit retreats onto a hexagon where it is out of range of both blue units. In that case the blue units advance, and both get into contact with the red unit. In the second situation, the red unit retreats onto a hexagon that is in range of one blue unit. In that case, the one unit opens fire, causing 50 point on the score. The second unit will advance into contact with the red unit, forcing it to retreat again. In the next situation where the red unit is in contact with only one blue unit, the unit gets destroyed (see Figure 23).

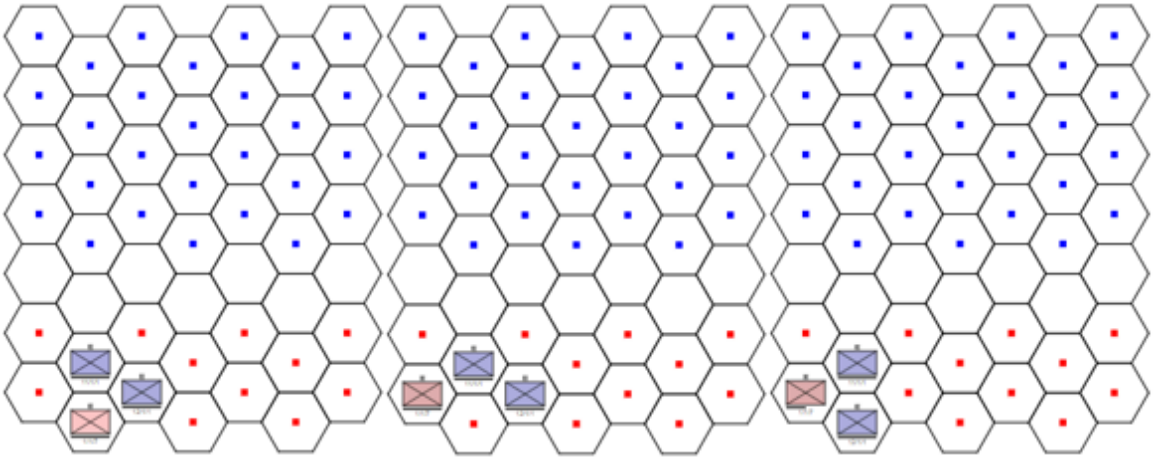

Figure 23. Red Forces Behavior in a Withdraw Situation

2. Results

For the first configuration ("fixed") of this scenario five blue agents are trained for 2,000,000 training steps. After training, each agent is tested in 10 full repetitions against a red force using the shootback AI. To measure the agent's training progression, the average achieved score and percentage of repetitions played with perfect behavior were collected at an increment of 100,000 training steps. Three out of the five agents started to show perfect behavior already after a few thousand training steps (see Figure 24). One agent adapted perfect behavior early in the training process but switched to imperfect behavior at around 600,000 training steps. Another agent quickly learned to attack with just one unit, but never learned to attack with both.

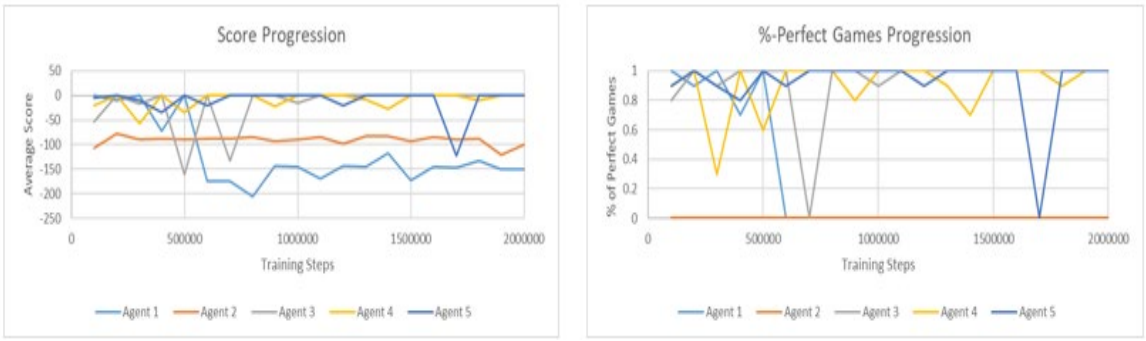

Figure 24. Two-versus-One Fixed Starting Positions Training Progression.

For the second configuration ("fixed formation") of this scenario, five agents are trained for 5,000,000 training steps. As with configuration 2, each agent is tested in 10 full repetitions against a shootback AI. One agent achieved stable, perfect behavior for the first time at around 2,000,000 steps. In general, all agents improved their average achieved score value over time, ending up with an average score value close to 0 after 5,000,000 training steps (see Figure 25). However, only one agent came into a position of winning all 10 repetitions with optimal behavior. Overall, the percentage of repetitions played with optimal behavior ranges between 40–100% when conducting more than 4,000,000 training steps.

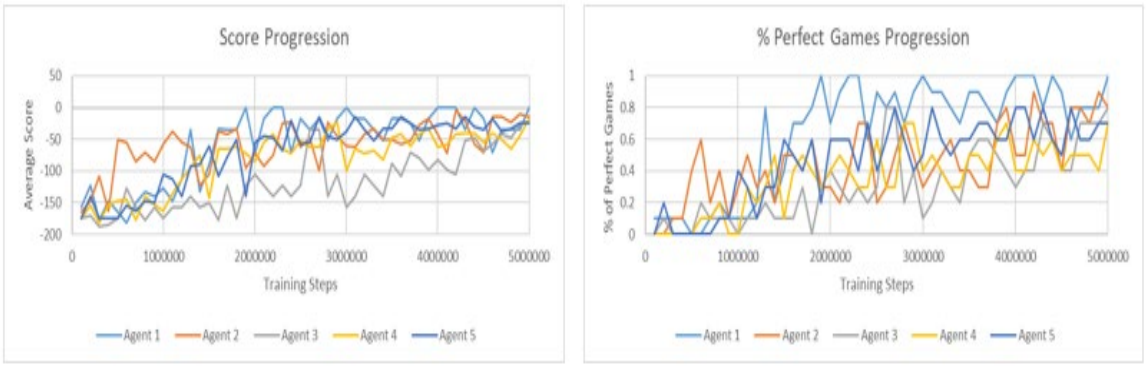

Figure 25. Two-versus-One Fixed Starting Formation Training Progression.

Similar to the second configuration, the agents in the third configuration ("random") were trained with 5,000,000 training steps. Over the whole training process, none of the agents managed to show perfect behavior in all of the played ten repetitions. As a result, the average achieved score always stayed below the value 0 (see Figure 26). Based on a visual assessment of the percentage of optimal played repetitions, it seems that above 4,100,000 training steps, all agents arrive at a state where they win at least 20% of the played games with optimal behavior. However, a training process going above the used 5,000,000 training steps might deliver more reliable data for this assessment.

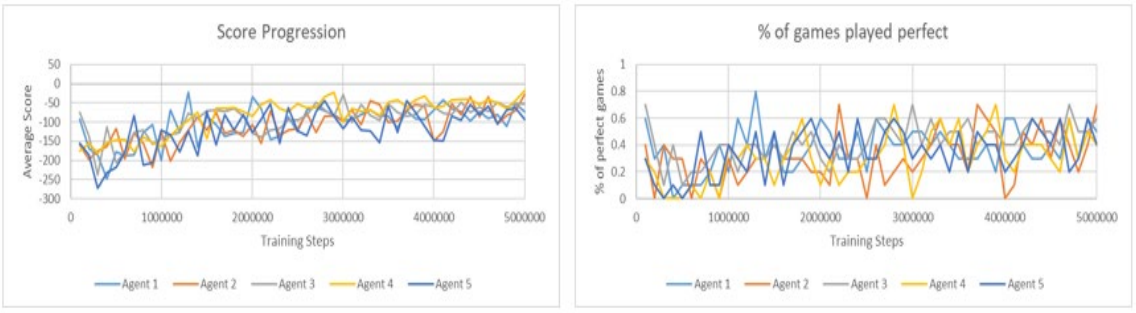

Figure 26. Two-versus-One Random Starting Formation Training Progression.

For the fourth configuration ("withdraw"), 10,000,000 training steps were used. Again, five agents were trained and evaluated in 10 repetitions against a withdraw AI at each increment of 1,000,000 training steps. Two agents achieved nearly optimal scores on average after 7,000,000 training steps and kept that performance for the remain of their training regimen. Two agents achieve a stable average score value of about -37.5 points, but optimal performance was never achieved. These agents demonstrated a behavior in which both units attacked in a straight line, with one blue unit directly following the other. The fifth agent started to improve his behavior at about 5,000,000 steps but never achieved am average score value comparable with the two best agents (see Figure 27).

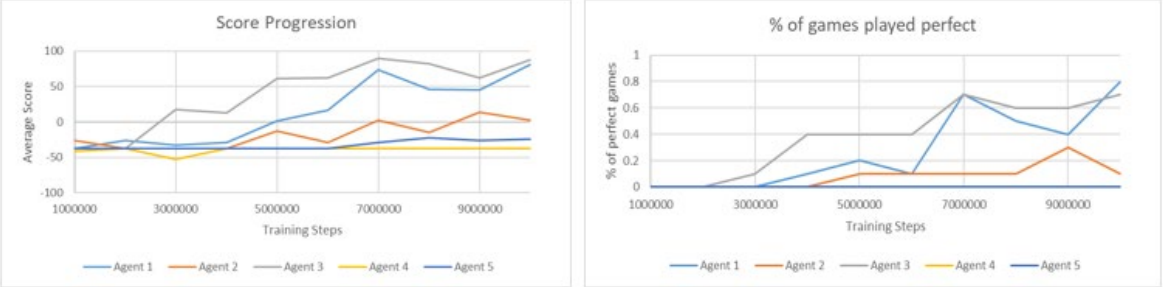

Figure 27. Two-versus-One Withdraw AI Formation Training Progression.

B. TWO-VERSUS-ONE SPATIAL INVARIANCE

1. Description

CNNs and their convolutional layers are specifically designed to achieve spatial invariance, meaning the neural network is capable of identifying key features in an image no matter where they are located. In this scenario, an agent will be evaluated on its ability to achieve spatial invariance by performing the same behaviors across the entire map after training on a fixed two-versus-one configuration. As shown in Figure 28, the agent will be trained on a map where all units are set up in fixed absolute positions. Once training is complete, the agent will then be evaluated on a map consisting of fourteen different setup positions. The setup up positions will start with setup 1 in the top left where the top blue unit is located in hexagon (0,0) and will scan left to right, top to bottom, ending with setup

14 where the top blue unit is in hexagon (6,2). The relative positioning of the units during the setup phase will be kept constant in the stack formations, and they will all be shifted randomly over the battlefield. The agent must perform the same massing behavior described in the previous scenario for all fourteen setup configurations to achieve optimal performance.

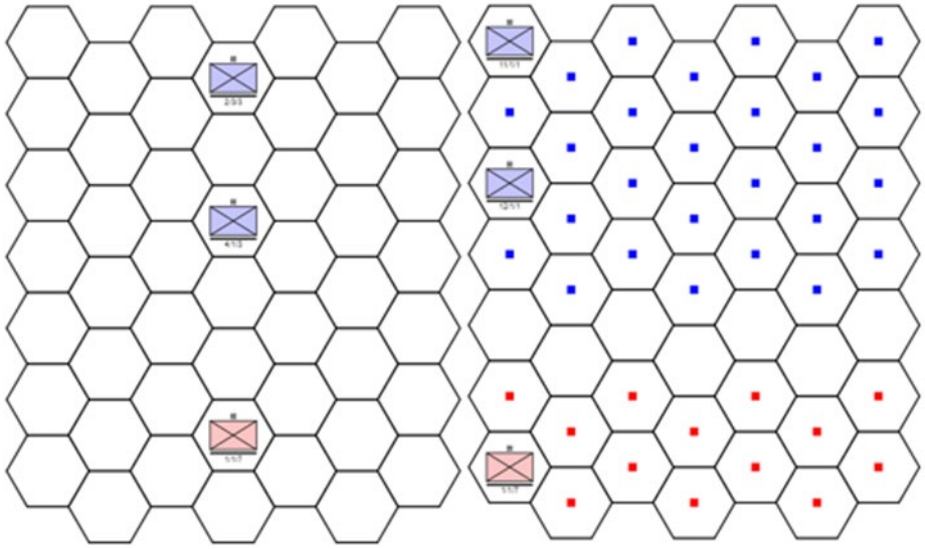

Figure 28. Two-versus-One Spatial Invariance Setup Configurations (left: training map / right: evaluation map)

To reduce the number of values being outputted by the neural network during training, an additional MaxPool2D layer was added. The standard CNN structure for this thesis typically outputs a 9-by-8 activation map. By inserting a 9-by-8 MaxPool2D layer after the third convolutional layer, the final activation map is reduced to only a single number (see Table 6). By minimizing the neural network's outputs, the agent's behaviors should remain the same throughout the battlefield as long as the same activation map value is achieved.

Table 6. Spatial Invariance CNN-Architecture

Layer	Input Channels	Output channels	Kernel size	Stride	Padding
1	3	32	(3,5)	(1,2)	2
2	32	64	(3,3)	(1,1)	1
3	64	64	(3,3)	(1,1)	1
MaxPool2D	-	-	(9,8)	(1,1)	-

2. Results

The neural network trained five separate agents for 2,000,000 training steps. After training, each agent was evaluated on its ability to demonstrate optimal performance on all fourteen possible setup configurations in the evaluation map. Ten complete repetitions were completed for each setup position and the total discounted reward was logged. The total discounted reward was the only performance value used for this scenario. To achieve optimal performance, the trained agent needs to receive a total discounted reward of 100 points.

All five agents' behaviors were identical (see Table 7). Each agent learned the optimal behavior on the center setup position (setup 6) in which it was trained, where both blue units entered hexagons adjacent to the red unit on the same turn and destroyed the red unit in two attacks. However, once the agents were shifted to other setup positions, their performances varied depending on which column they started in.

Table 7. Two-versus-One Spatial Invariance Agent Evaluation at 2,000,000 Training Steps

	Average Total Discounted Reward													
	Setup 1	Setup 2	Setup 3	Setup 4	Setup 5	Setup 6	Setup 7	Setup 8	Setup 9	Setup 10	Setup 11	Setup 12	Setup 13	Setup 14
Agent 1	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 2	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 3	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 4	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 5	0	100	0	100	0	100	0	0	100	0	100	0	100	0

For all setup positions in odd columns, the agents' tactics mimicked what was shown on the training setup positions as shown in Figure 29.

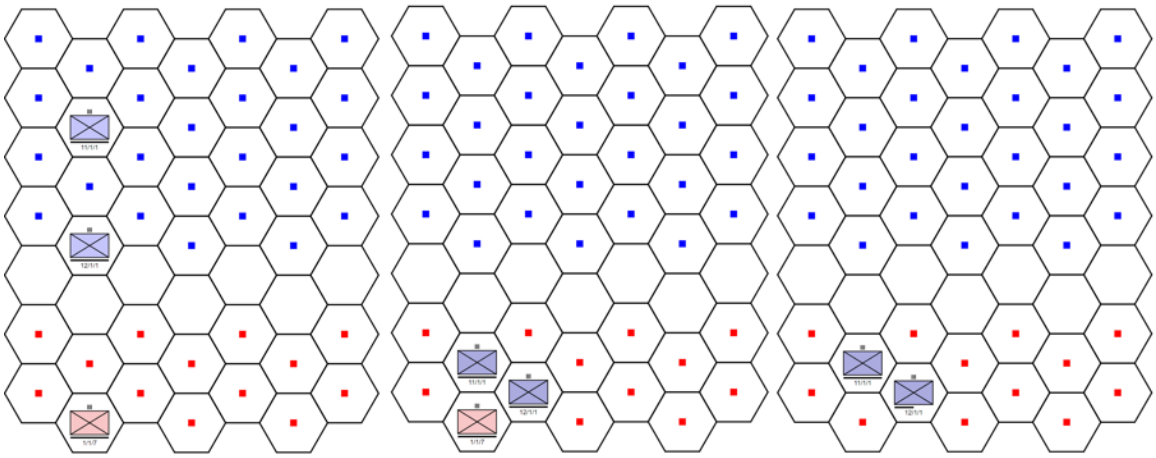

Figure 29. Two-versus-One Spatial Invariance Odd Column Demonstration

When the agents were evaluated in setup positions starting in even columns though, only random movements were shown by the trained agent for the entirety of the repetition (see Figure 30). It was evident that the agents did not recognize that current state and did not know what behaviors needed to be performed.

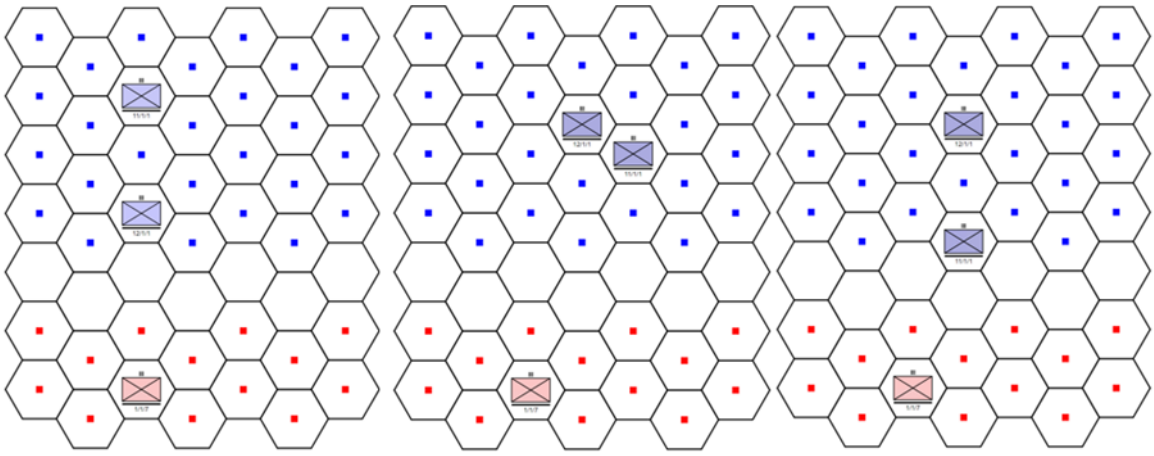

Figure 30. Two-versus-One Spatial Invariance Even Column Demonstration

C. TWO-VERSUS-TWO

1. Description

Increasing the number of units from the first scenario, this scenario is comprised of a two-versus-two engagement where two blue units attack two red units. The red force will remain stationary and solely use the shootback AI for its combat behaviors. As shown in Figure 31, two different configurations will be evaluated for this scenario. The first configuration entails both the red and blue forces having fixed starting positions that remain constant during each iteration. In the second configuration, the blue force's starting position will vary during each repetition. Each unit has a set of six possible starting locations of which one is randomly chosen at the start of each repetition. However, the red force will continue to remain in a fixed starting position located at the center of the map.

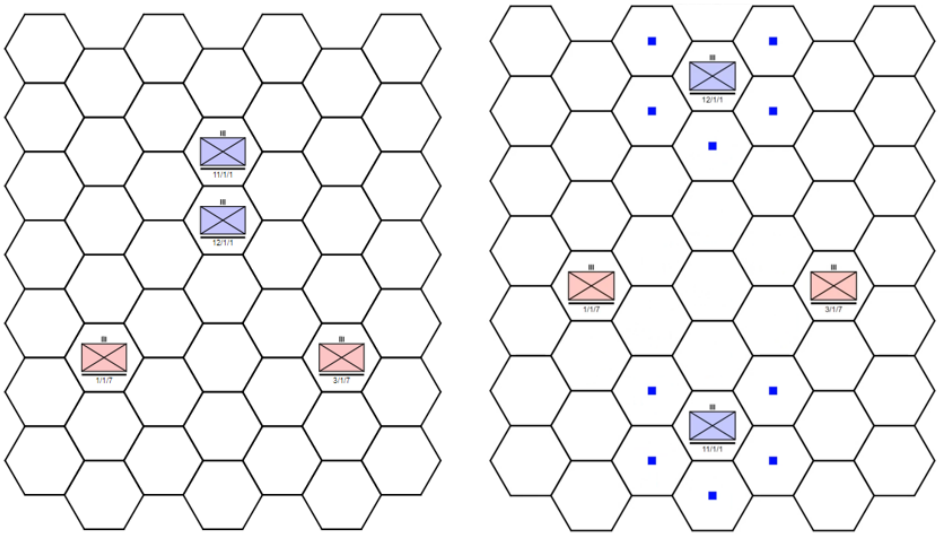

Figure 31. Two-versus-Two Setup Configurations (left: Fixed Positions / right: Multiple-Starting-Positions).

For the trained agent to achieve optimal performance in this scenario, it must successfully learn how to conduct two coordinated attacks. The blue units must first start by simultaneously attacking one of the red units. Once that red unit has been destroyed, both blue units need to navigate to the remaining red unit and conduct another simultaneous attack. If the two blue units accomplish successful coordination, then the resulting blue force will have only

one unit remaining with either 50 or 75 heath depending on who the red units randomly decide to attack. The optimal behavior will remain the same no matter the configuration.

Similar to the two-versus-one scenario, Atatl's scoring system in combination with the blue agent's discounted reward will be used to evaluate the blue agent's performance. The optimal score the blue agent is seeking to achieve is either -50 or -100 points, and a discounted reward of either 150 or 175 points. After conducting the first simultaneous attack, the score will be 0 due to the trained agent having one unit at full health and one unit at 50 health and the red force having only one unit with full health. As both blue units are still alive, the trained agent will receive the full reward of 100 points for the first attack.

During the second attack, if the red unit decides to attack the blue unit with 50 health, that unit will be destroyed, but the other blue unit can attack using its full strength to inflict 50 damage on the first turn and 25 damage on the second turn. That unit's health will also reduce by 25 health from the red attack. With only one blue unit alive, the reward will now be discounted by 50 percent, so the trained agent will only receive 50 points for the destruction of the red unit. The final score of this iteration will be -50, and the trained agent will receive a discounted reward of 150 points. If the remaining red unit decides to attack the blue unit with full health instead, both blue units will now be left with 50 health. During the trained agent's initial attack, it will receive the full 50 points for the damage inflicted as both units are still alive but will have one unit destroyed during the red unit's attack. In the final turn, the final blue unit will destroy the red unit and receive a discounted reward of 25 points. The final score for this iteration will be -75 points, with a total discounted reward of 175 points. There is an even distribution between the two possibilities this scenario can unfold.

Table 8. Two-versus-Two Scenario Optimal Behavior Performance Values.

Configuration	Optimal Blue Strength	Optimal Total Discounted Reward	Optimal Score
Fixed	50/75	150/ 175	-50/ -100
Multiple-Starting	50/75	150/175	-50/ -100

2. Results

For each configuration in this scenario, five different blue agents were trained at 2,000,000 and 3,000,000 training steps, respectively. Each agent was evaluated with 100 repetitions at every increment of 100,000 training steps. Each repetition completes a full engagement between the blue and red forces, and the resulting score is recorded. The achieved score and discounted reward for each agent at the specific training step is shown in Figure 32. The optimal values, as described in Table 8, are visualized as black lines.

All five trained agents were able to achieve the required score and discounted reward needed for optimal performance by the conclusion of their training. While four out of the five agents reached optimal performance before 500,000 training steps, Agent 2 required 1,800,000 training steps before learning the optimal behaviors. The slight differences in the input features based on which blue unit received damage during the first simultaneous attack caused the neural network difficulty understanding the appropriate tactics needed for the second attack. While all of the other agents did achieve optimal behaviors much earlier, there were brief instances of instability where their average score and discounted reward dropped below the optimal threshold. Each of the agents were able to make the quick corrections needed to get back on track.

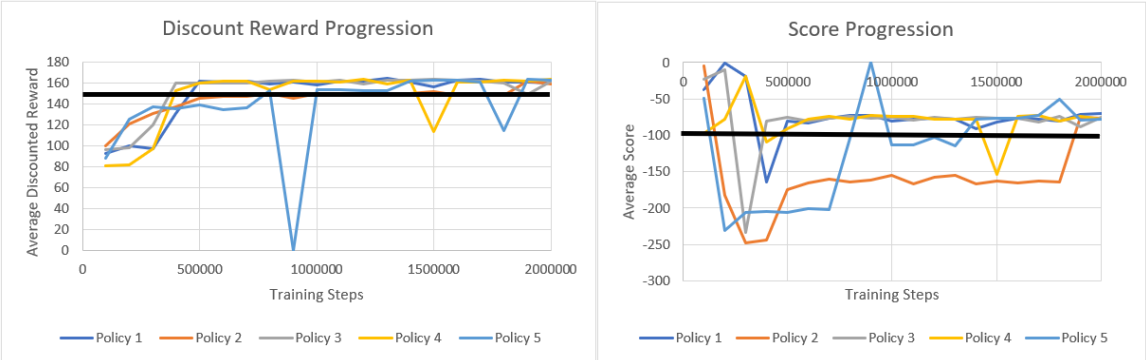

Figure 32. Two-versus-Two Fixed Starting Positions Training Progression.

After training, all of the agents yielded an average score close to -75 and an average discounted reward of near 160. This score validated that no matter what blue unit was

initially attacked, the blue agent displayed the appropriate tactics needed to destroy the red force while receiving the fewest number of casualties. Figure 33 displays the resulting tactics for a single repetition that resulted in a resulting score of -50 and a total discounted reward of 150, with one blue unit remaining with 75 health. The tactics show the trained agent massed both units simultaneously on the left red unit, reorienting themselves after the first unit is destroyed, then conducted another simultaneous attack on the right red unit. Through understanding the importing of massing its forces, the blue force effectively destroyed each of the red units.

Table 9. Two-versus-Two Fixed Starting Positions Agent Evaluation at 2,000,000 Training Steps.

	Agent 1	Agent 2	Agent 3	Agent 4	Agent 5
Avg. Discounted Reward	160.3 +/- 1.2	159.3 +/- 1.8	162.0 +/- 1.3	163.8 +/- 1.3	162.6 +/-1.3
Avg. Score	-70.5 +/- 2.5	-76.0 +/- 4.1	-75.8 +/- 2.7	-77.5 +/- 2.5	-77.5 +/- 3.1
Optimal Performance	Yes	Yes	Yes	Yes	Yes

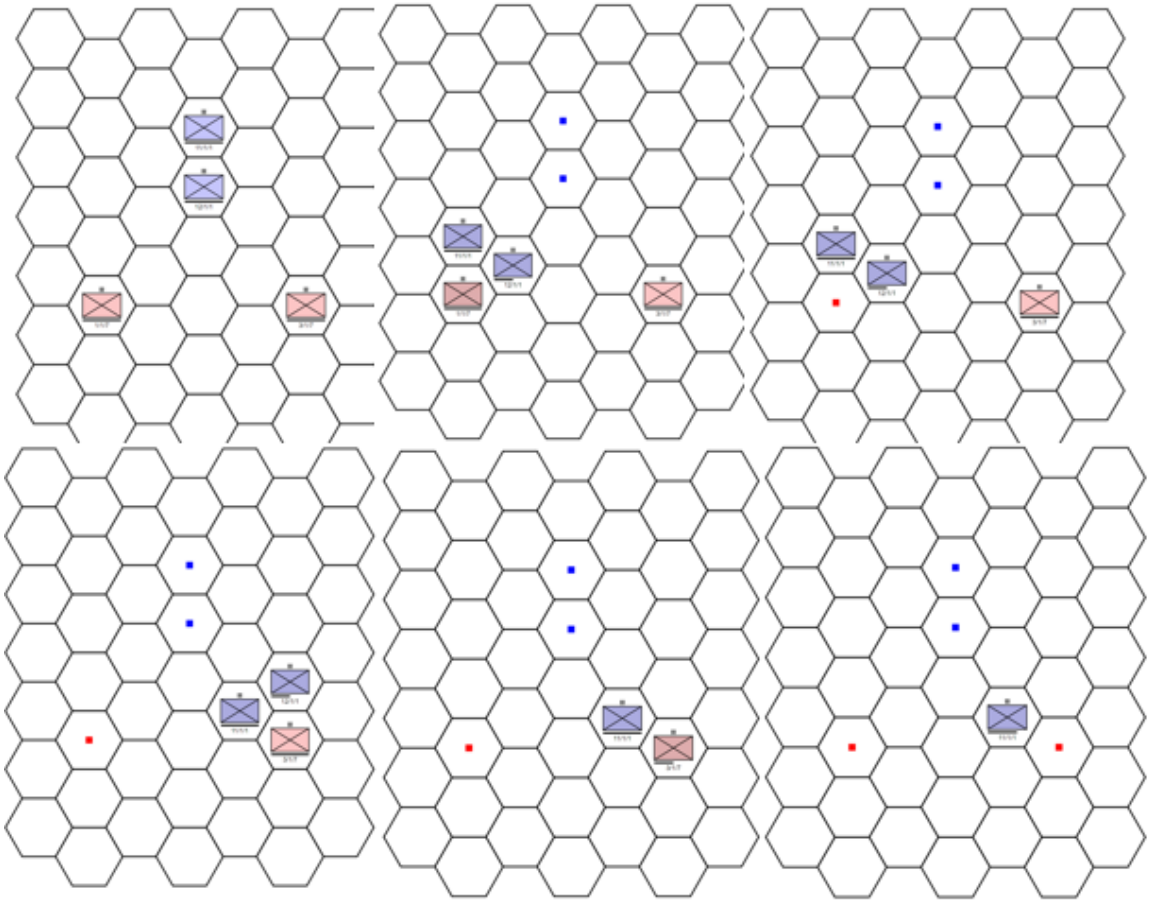

Figure 33. Two-versus-Two Fixed Starting Position Optimal Performance Demonstration.

Due to the increased complexity when introducing multiple starting positions in the 2v2-MSP-configuration, an additional 1 million training steps were awarded to each agent when training. However, the method for evaluating each agent remains the same. Unlike in the previous configuration, all five trained agents achieved the average score needed for optimal performance at similar times, but as expected, they required an additional 300,000 training steps. The majority of the training progression remained stable, but each agent also experienced a brief portion of instability where corrections needed to be made. Overall, the neural networks effectively learned the optimal behaviors and retained the appropriate tactics throughout the training.

Figure 34. Two-versus-Two Multiple Starting Positions Training Progression.

After 3,000,000 training steps, each trained agent achieved an optimal average score near -75 points, and an average discounted reward close to 160 points. While the starting positions may have differed, the tactics displayed remain the same as the fixed position configuration tactics. The results show the blue units coordinating their first attack where both units move into the left red unit's range on the same turn. After the left red unit is destroyed, the blue units reorganize, then conduct another simultaneous attack on the right red unit. Figure 35 displays the results when the outcome has a blue unit with 50 strength, a final score of -100 points, and a total discount reward of 175 points.

Table 10. Two-versus-Two Multiple Starting Positions Agent Evaluation at 3,000,000 Training Steps.

	Policy 1	Policy 2	Policy 3	Policy 4	Policy 5
Average Score	-74.5 +/- 2.5	-76.0 +/- 4.5	-78.0 +/- 2.5	-79.8 +/- 3.0	-79.8 +/- 3.0
Average Discounted Reward	162.2 +/- 1.3	163.0 +/- 1.3	164.0 +/- 1.3	160.9 +/- 1.6	162.3 +/- 1.4
Optimal Performance?	Yes	Yes	Yes	Yes	Yes

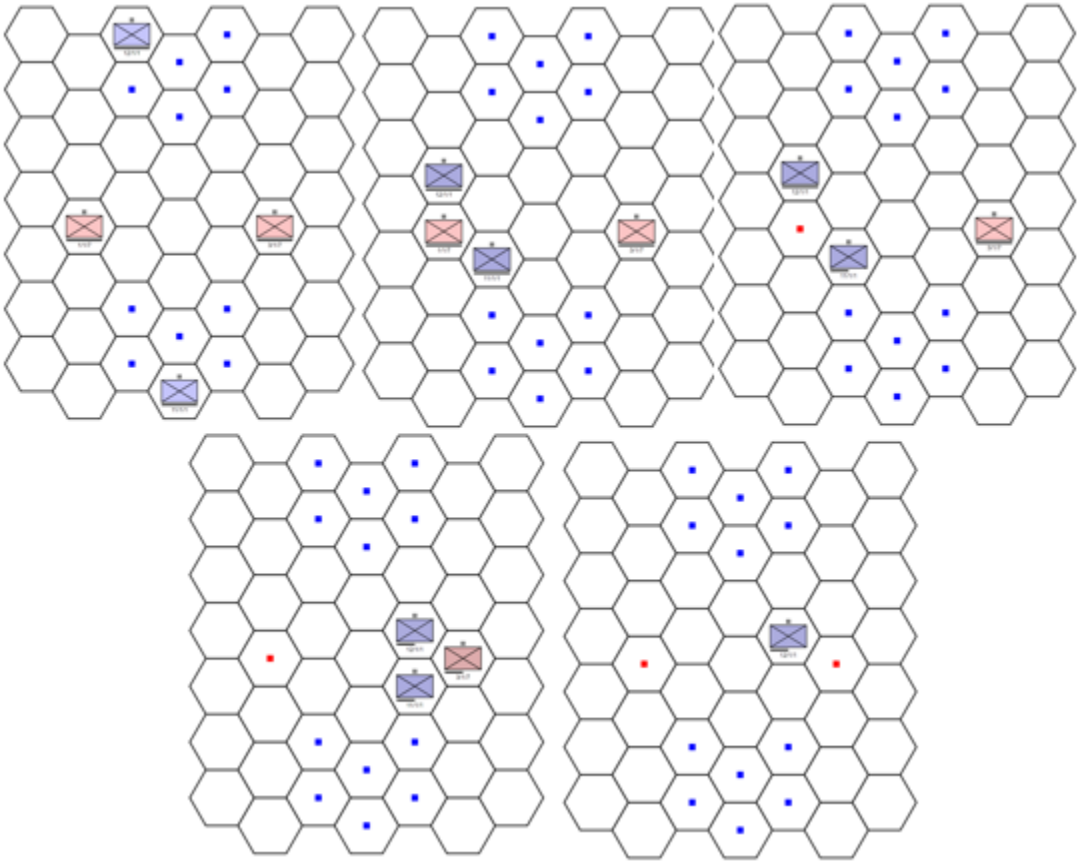

Figure 35. Two-versus-Two Multiple Starting Position Optimal Performance Demonstration

D. THREE-VERSUS-TWO

1. Description

Further expanding on the two previous scenarios, this scenario encompasses a three-versus-two engagement, where three blue units attack two red units. To replicate Boron's three-versus-two scenario [6] accurately where one red unit was a lower echelon than the other red unit, the red force in this scenario will have one red unit with 25 reduced strength points at the start of each iteration. The red force's behavior will remain the same, where the units are stationary throughout the engagement and will utilize the shootback AI for its combat behaviors. The neural network will train the blue force on two different scenario configurations, as shown in Figure 36. The first configuration entails fixed starting positions for both the blue and red forces. For the second configuration, the red force will

start in the same location, but the blue force will be distributed throughout three separate regions located in the northwest, northeast, and southern portions of the map. In each region, each unit has a set of six possible starting locations of which one is randomly chosen at the start each repetition.

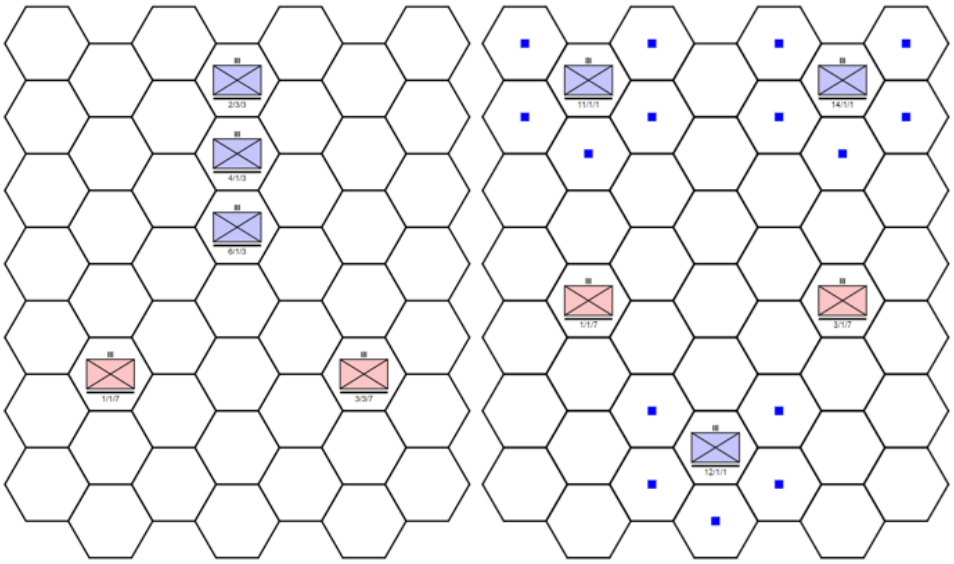

Figure 36. Three-versus-Two Setup Configurations (left: Fixed Positions / right: Multiple-Starting-Positions)

In Boron's three-versus-two scenario [6], the agent's performance was evaluated on its ability to exhibit both economy of force and massing through utilization of a discount factor. If a discount factor higher than 0.98 was used, the agent attacked using massing behaviors where all three blue units destroyed the stronger red unit simultaneously before moving toward the other weaker unit. If using a discount factor lower than 0.98, the agent looked to win the engagement as quickly as possible by having two units simultaneously attack the stronger red unit while the other blue unit attacked the other red unit alone. Once the stronger red unit was destroyed, the two other blue units would mass on the remaining blue unit. While this thesis aims to recreate all of Boron's scenarios utilizing a CNN instead of an MLP neural network, due to the difference in how combat is modeled in Atlatl, economy of force will be the only combat behavior evaluated for the blue force. For infantry engagements in Atlatl, most cases only require two successful attacks to destroy

another unit. If the third blue unit were to try to participate in that engagement to show a massing behavior as seen in Boron's work, the red unit will be destroyed before the it has an opportunity to attack.

For this scenario, optimal performance can only be achieved if the trained agent is able to properly learn how to use economy force to engage both red units simultaneously. One blue unit must start by attacking the weaker red unit on the right, while the other two blue units simultaneously attack the left red unit. If successful economy of force behavior is shown, then the blue agent will have one unit with full strength, one unit with 75 strength, and one unit with 50 strength. The trained agent's performance will be evaluated using Atlatl's scoring system and their total discount reward.

The optimal performance scores the trained blue agents need to achieve are the same for both configurations and was calculated through manual iterations of the scenario (see Table 11). When the single blue unit attacks the weaker red unit on the right, it will initially take 25 damage before destroying the red unit with 75 damage, causing the score to be 0. When the two blue units attack the red unit on the left, one blue unit will receive a 50-damage loss before both of them destroy the red unit with 100 damage, resulting in a score of 0. Due to all three blue units being alive at the end of the scenario, the trained agent will receive the full reward of 175 points.

Table 11. Three-versus-Two Scenario Optimal Behavior Performance Values

Configuration	Optimal Blue Strength	Optimal Total Discounted Reward	Optimal Score
Fixed	225	175	0
Multiple-Starting	225	175	0

2. Results

The neural network trained five different agents for each configuration in this scenario. With the increased complexity when adding an additional blue unit, additional training time was awarded. 3,000,000 training steps was used for the fixed starting position configuration, and 4,000,000 was used for the multiple starting positions configuration.

As shown in Figure 37, four out of the five trained agents learned the behaviors needed to achieve the optimal score and discounted reward for the fixed starting position configuration. Most of the agents learned the tactics needed to destroy the red unit with full health almost immediately, as depicted by the average discounted reward of 100 at 100,000 training steps. However, it was not until approximately 2,000,000 training steps before most of the agents learned how to use the additional blue unit to attack the remaining red unit. Once the agent achieved the optimal score and discounted reward, their behavior remained relatively stable for the remainder of the training. For the agent that did not achieve optimal performance, Agent 2, proper economy of force tactics was never learned. Rather than splitting its forces and using all three units to attack each red unit simultaneously, the same two blue units destroyed the first red unit to attack the other red unit then, leaving one blue unit utterly unused for the entirety of the iteration. As the same two units were used for both attacks, the trained agent risked having the complete destruction of one blue units which would results in 33.3% lower discounted reward.

Figure 37. Three-versus-Two Fixed Starting Positions Training Progression

When evaluating the trained agents at 3,000,000 training steps, all agents besides Agent 2 achieve an average score close to 0 and an average discounted reward near 173 points (see Table 12). These scores show that most agents understood the importance of distributing their forces to appropriately attack both red units simultaneously, as described previously and shown in Figure 38.

Table 12. Three-versus-Two Fixed Starting Positions Agent Evaluation at 3,000,000 Training Steps.

	Agent 1	Agent 2	Agent 3	Agent 4	Agent 5
Avg. Discounted Reward	0.0 +/- 0.0	-15.5 +/- 4.6	-1.5 +/- 1.1	3.5 +/- 2.2	-4.8 +/- 2.9
Avg. Score	175.0 +/- 0	131.8 +/- 3.8	172.8 +/- 1.3	172.0 +/- 1.5	172.8 +/- 1.3
Optimal Performance	Yes	No	Yes	Yes	Yes

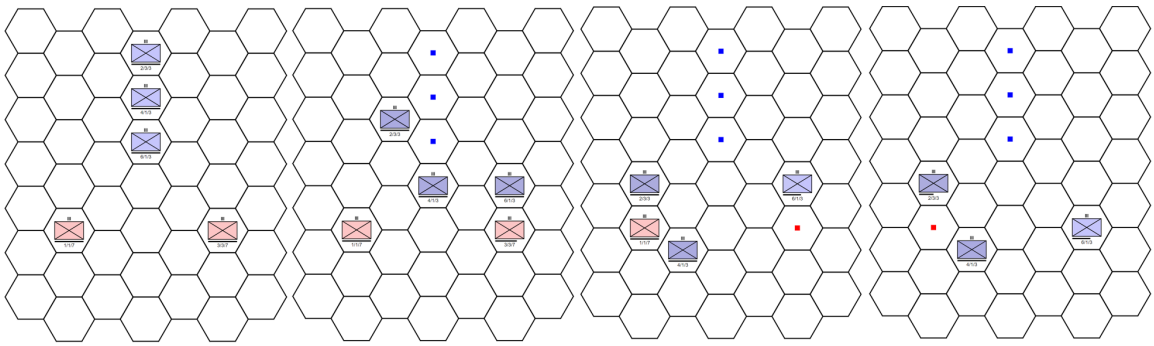

Figure 38. Three-versus-Two Fixed Starting Position Optimal Performance Demonstration.

Unlike the previous configuration, all agents trained on the multiple starting position configuration achieved the optimal score and discount reward needed for optimal performance by the end of their training regimen. Surprisingly, the neural network had better performance when training the agents on the multiple starting position configuration than the stack configuration. Rather than starting in a stack formation that required a high level of coordination between the units during initial turns, the units could move freely during their initial turns. The only coordination that needed to occur between the blue units was for them to occupy adjacent hexagons to the red unit in the same turn. The more straightforward tactical approach allowed the agents to learn the optimal quicker than the fixed configuration.

The agents learned how to split their forces and coordinate a simultaneous attack early on during their training. As shown in Figure 39, it only took four out of five agents 500,000 training steps to achieve the optimal performance values, vise the 2,000,000 in the fixed configuration. While the performance was initially better, there were some occurrences of instability for several starting positions where the trained agents did not coordinate their attack on the stronger red unit appropriately. This behavior resulted in two units with 50 health and one unit with 75 health, which cause the final score to be lower without affecting the discounted reward (see Table 13).

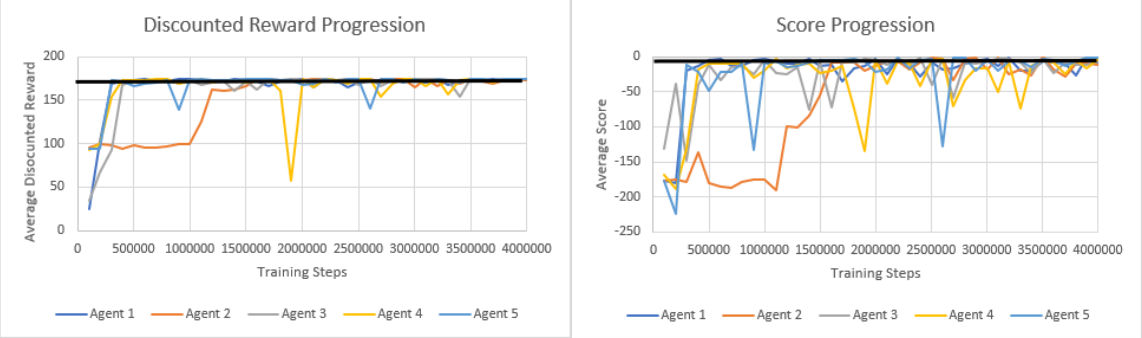

Figure 39. Three-versus-Two Multiple Starting Positions Training. Progression

Figure 40 demonstrates the optimal performance achieved by one of the trained agents. The units positioned in the northwest and south starting regions initially start by moving toward the red unit with full health, while the blue unit starting in the northeast region moves toward the red unit with reduced health. Simultaneously, all three blue units coordinate an attack where each red unit is destroyed, leaving one blue unit with full health, one with 75 health, and one with 50 health.

Table 13. Three-versus-Two Multiple Starting Positions Agent Evaluation at 4,000,000 Training Steps.

	Agent 1	Agent 2	Agent 3	Agent 4	Agent 5
Avg. Discounted Reward	7.8 +/- 2.9	-10.5 +/- 3.83	-6.5 +/- 1.8	-2.0 +/- 1.2	-0.5 +/- 0.5
Avg. Score	173.5 +/- 0.8	174.8 +/- 0.2	173.4 +/- 0.9	174.8 +/- 0.2	174.8 +/- 0.3
Optimal Performance	Yes	No	Yes	Yes	Yes

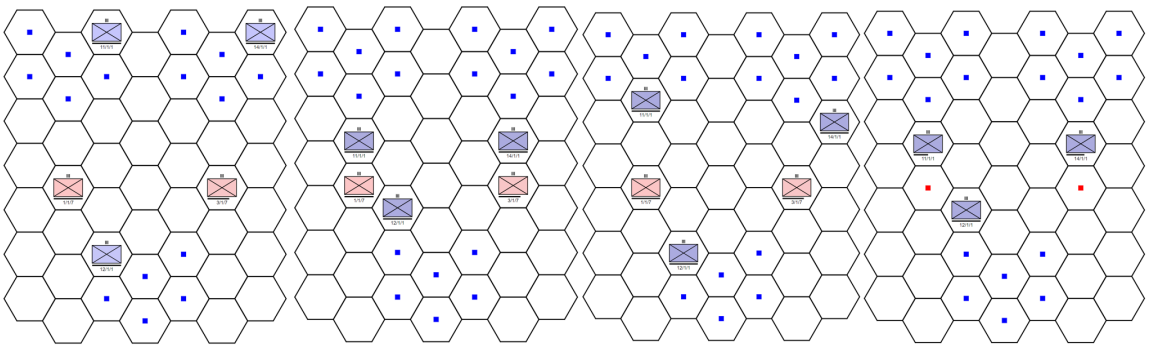

Figure 40. Three-versus-Two Multiple Starting Position Optimal Performance Demonstration.

E. URBAN TERRAIN

1. Description

The training scenario involved a two-versus-one configuration where two blue units attack a single stationary red unit. Expanding on previous two-versus-one configurations, the complexity of the training environment was increased by introducing a single urban terrain hexagon in the southwest portion of the map (see Figure 41). The urban terrain hexagon rewards a fixed bonus to a force if one of their units occupies the hexagon at the end of each turn. To continuously receive the urban terrain's bonus after initial occupation, one unit in that force must remain in the hexagon; otherwise, no bonus will be received.

For this scenario, agents will be trained on three different urban terrain hexagon values (0, 20, and 40), and the responding behaviors will be analyzed.

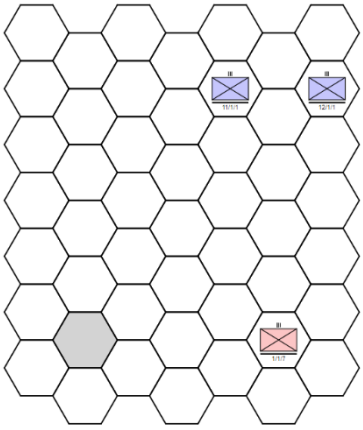

Figure 41. Two-versus-One Urban Terrain Scenario Setup Configuration.

When examining behaviors for the scenario with the urban terrain hexagon valued at 0 points, the optimal behavior requires the trained agent to mass its force on the red unit with complete disregard for the urban terrain hexagon. Similar to the previous two-versus-one configuration, the final score will be zero, and the total discounted reward will be 100. For the configuration with urban terrain valued at 60 points, optimal performance is achieved when both blue units bypass the red unit and occupy the urban terrain using the shortest and quickest possible route. After initial occupation on the 11th turn, the blue force must continue to have one unit remain in the hexagon for the rest of the iteration, resulting in a final score and total discount reward of 600 points.

For the second configuration, when the urban terrain hexagon is valued at 20 points, more robust behaviors are required from the trained agent to achieve optimal performance. Both blue units must initially start by moving toward the red unit and occupy hexagons adjacent to the red unit on the same turn, allowing the red unit to attack first. There are two possible courses of action for optimal performance, both of which depend on the blue unit initially attacked. If the right blue unit is attacked first, both blue units will remain in a

position to attack and destroy the red unit. Once it has been destroyed, the left blue unit needs to navigate to the urban terrain hexagon and occupy it on the 15th turn. However, if the left blue unit is attacked first, it will immediately start heading toward the urban terrain hexagon instead of participating in the attack. Due to the left blue unit being attacked first, the right blue unit with full strength now has the attacking advantage and can destroy the red unit independently. While the strength of the blue force and the score will be lower at the end of the iteration, the reward for occupying the urban terrain hexagon on the 13th turn versus the 15th outweighs the reduced combat reward. The specific values needed for optimal performance are outlined in Table 14.

Table 14. Two-versus-One Urban Terrain Scenario Optimal Behavior Performance Values.

Urban Terrain Value	Optimal Blue Strength	Optimal Red Strength	Optimal Total Discounted Reward	Optimal Score
0	150	0	100	0
20 **(Left/ Right Attack)**	125/ 150	0	220/ 260	110/ 120
60	200	100	600	600

2. Results

Five separate agents were trained at 2,000,000 training steps against a red force using the shootback AI for each urban terrain hexagon value. In total, there were 15 agents trained for this scenario.

By adjusting the urban terrain hexagon value, the agents successfully learned three district behaviors that maximize the total discounted reward received for each configuration. As expected, when the urban terrain value was 0 or 60 points, all agents could easily learn the optimal behaviors quickly, as no other behaviors resulted in a final discounted reward near the value of the optimal behavior. When the urban terrain was valued at 20 or 60 points, all five agents learned the optimal behavior in less than 100,000 training steps and retained those behaviors for the duration of training. When evaluated after 2,000,000 training steps, each agent for those two urban terrain achieved the optimal score during each repetition (see Table 15). However, when the urban terrain was valued

at 20 points, only three of the five agents were able to learn the optimal behaviors. The other two agents got stuck in a local optimum where they completely ignored the red unit and went straight to occupying the urban terrain hexagon. The two blue agents never learned that destroying the red unit before occupying the urban terrain hexagon would result in their total discounted reward approximately 60-points greater.

Table 15. Two-versus-One Urban Terrain Agent Evaluation at 2,000,000 Training Steps.

	Urban Value	Agent 1	Agent 2	Agent 3	Agent 4	Agent 5
Avg. Score	*0*	0.0 +/- 0.0	0.0 +/- 0.0	0.0 +/- 0.0	0.0 +/- 0.0	0.0 +/- 0.0
Avg. Disc. Reward	*0*	100.0 +/- 0.0	100.0 +/- 0.0	100.0 +/- 0.0	100.0 +/- 0.0	100.0 +/- 0.0
Avg. Score	*20*	239.6 +/- 2.0	180.0 +/- 0.0	243.6 +/- 2.0	180.0 +/- 0.0	238.8 +/- 2.0
Avg. Disc. Reward	*20*	115.1 +/- 0.5	80.0 +/- 0.0	114.1 +/- 0.5	80.0 +/- 0.0	115.3 +/- 0.5
Avg. Score	*60*	600.0 +/- 0.0	600.0 +/- 0.0	600.0 +/- 0.0	600.0 +/- 0.0	600.0 +/- 0.0
Avg. Disc. Reward	*60*	600.0 +/- 0.0	600.0 +/- 0.0	600.0 +/- 0.0	600.0 +/- 0.0	600.0 +/- 0.0

Figure 42 demonstrates the behaviors displayed by the trained agents during their evaluations when the urban terrain hexagon was valued at 0 and 60 points. When valued at 0 points, the blue agent demonstrated the same behavior previously seen in the two-versus-one scenario. Both units coordinated their attack, so they occupy the hexagons adjacent to the red unit on the same turn, receive minimal casualties, then destroy the red unit on the following turn. As the urban terrain hexagon provides no bonus for occupation, the trained agent remained in place following the destruction of the red unit. When the urban terrain hexagon value was worth 60 points, the trained agent decided to bypass the red unit altogether and head straight to the urban terrain. If the trained agent decided to attack the red unit then occupy the urban terrain hexagon, the maximum discounted reward it could have received was 340 points, 160 points less than the optimal behavior.

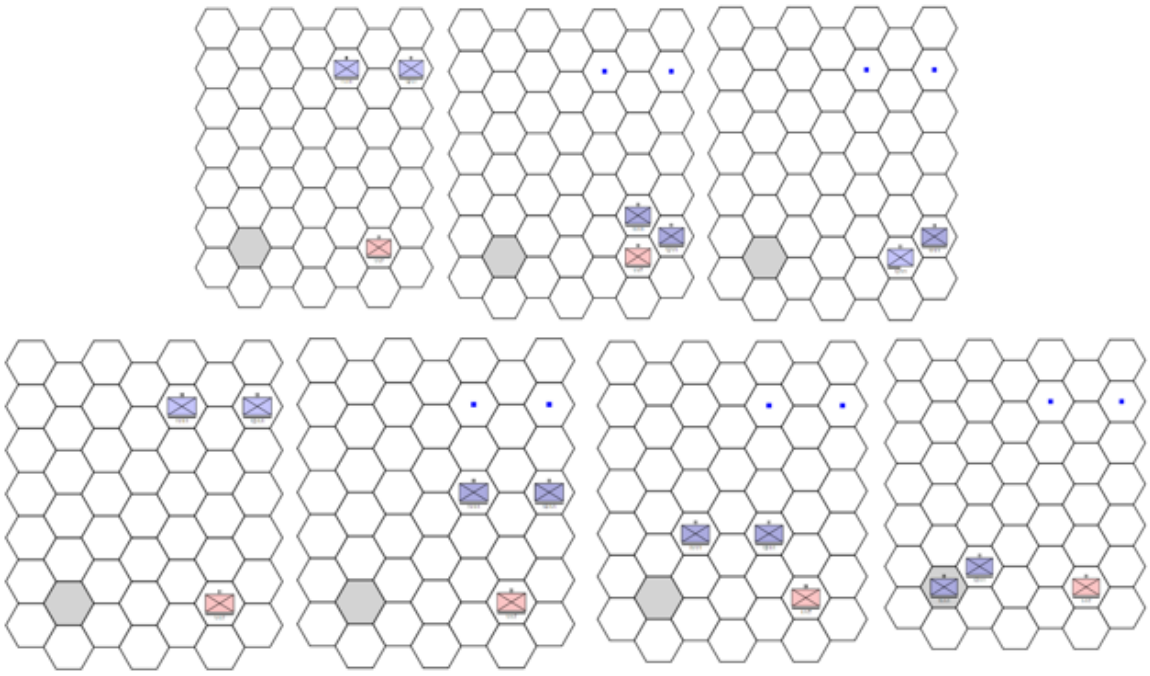

Figure 42. Two-versus-One Urban Terrain Value 0 Points (top) and 60 Points (bottom) Optimal Performance Demonstration.

While observing each agent's training progression when the urban terrain hexagon was valued at 20 points, it showed how critical the exploration in the first 200,000 training steps was for the agent's success. Each agent reached its highest or close to its highest discounted reward value that it would receive throughout the entire training regimen early on in their training. Due to the structure of PPO, the agent's exploration rate is at its highest during the initial training steps but then dramatically decreases as the training progresses. Due to this structure, the exploration rate is too low after 200,000 training steps for two of the agents to find another behavior outside of their local optimum, resulting in them never learning the optimal behaviors. However, the other three agents learned the optimal behavior before 200,000 training steps and retained it throughout the entire training regimen (see Figure 43).

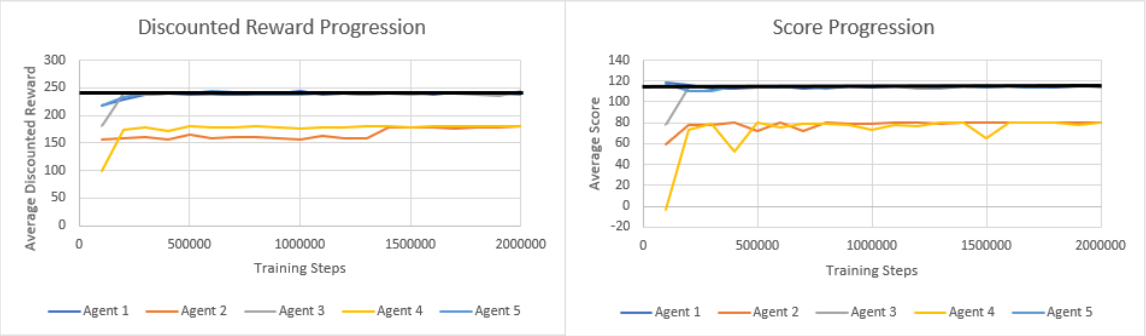

Figure 43. Two-versus-One Urban Terrain Value 20 Training Progression.

The optimal behaviors shown by Agents 1, 3, and 5 during their evaluation when the urban terrain hexagon is valued at 20 points is presented in Figure 44. The two demonstrations reveal how the trained agent's behavior differs based on which unit is randomly attacked by the red unit. If the left blue unit is attacked first, the left blue unit immediately travels toward the urban terrain and allows the right blue unit to destroy the red unit on its own. The iteration ends with the left blue unit occupying the urban terrain hexagon with 50 health and the right blue unit left with 75 health after destroying the red unit. However, when the red unit attacks the right blue unit, the left blue unit must help with the attack otherwise, the right blue unit will be killed. Once both blue units attack and destroy the red unit, the left blue unit advances toward and occupies the urban terrain hexagon.

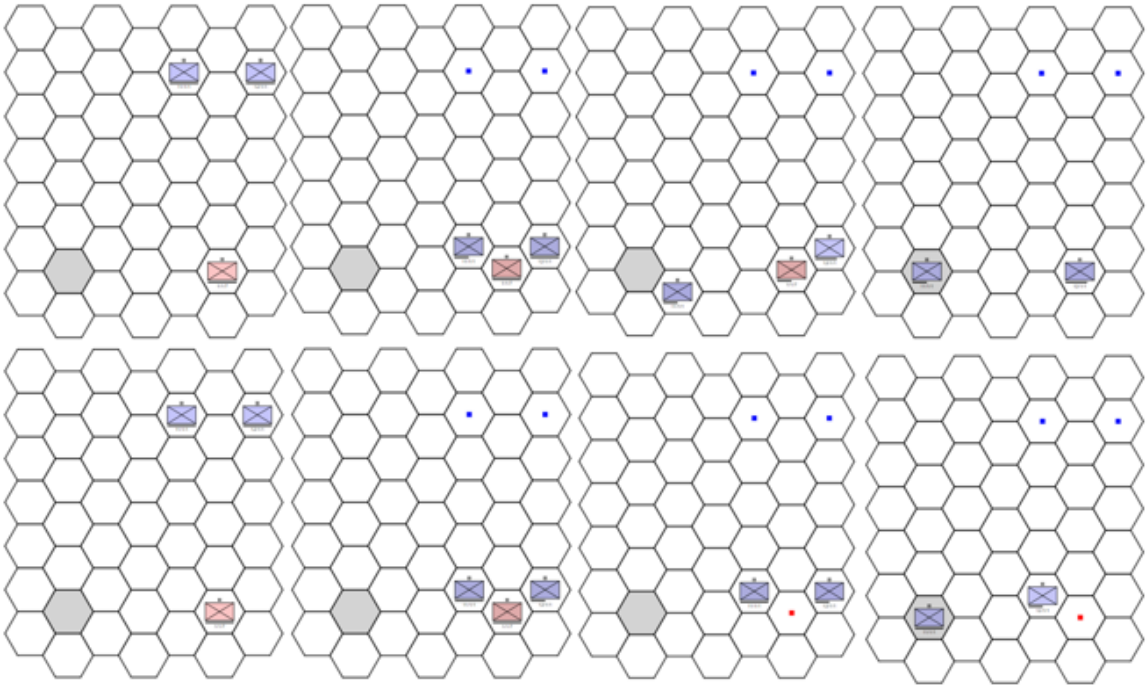

Figure 44. Two-versus-One Urban Terrain Value 20 Optimal Performance Demonstration (top: attack left first/ bottom: attack right first)

F. CHANGING URBAN TERRAIN LOCATION

1. Description

After manipulating an agent's behavior through altering the value of an urban terrain hexagon, this scenario will now look to observe the effect on an agent's behaviors when the urban terrain hexagon is moved around the map. In this scenario, an agent will be trained on five separate maps that will rotate consecutively after each training iteration of the scenario. When the agent starts its training, it will start on the first map. After 20 turns and that iteration is complete, the scenario will reset with the subsequent map loaded. This process will continue until the agent reaches its maximum training steps.

In each map, the units are placed in the identical location. The only thing that changes between each map is the location of the urban terrain hexagon. As seen in Figure 45, the five urban terrain locations include a center, northern, eastern, southern, and western position. The urban terrain hexagon rewards a fixed bonus of 24 points to a force

if one of their units occupies the hexagon at the end of each turn. As in the previous scenario, the unit must remain in the hexagon to continuously receive the additional bonus in the following turns.

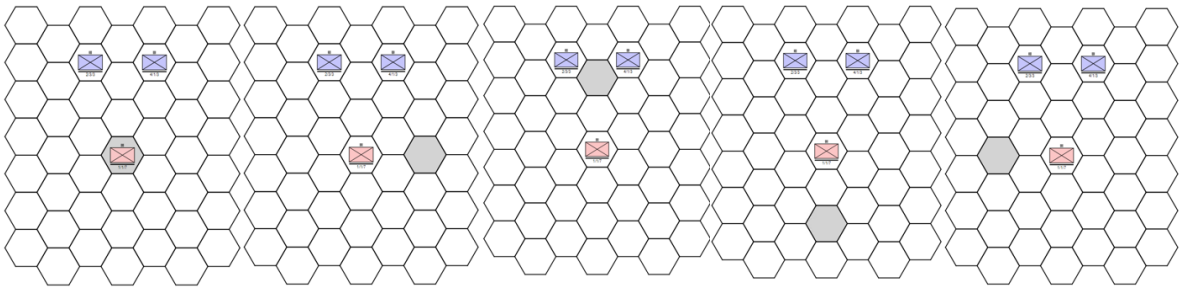

Figure 45. Changing Urban Terrain Location Maps (left to right: maps 1 to 5).

To account for the different urban terrain hexagon location in each of the maps, the neural network is provided an additional input that denotes the location of the urban terrain. Similar to the other input layers, a 1.0 is used to represent the hexagon as urban terrain and a 0.0 as clear terrain. As illustrated in Figure 46, the fourth array represents how the urban terrain location is represented as an input. Since the urban terrain is located in the (4,6) hexagon, there is 1.0 denoted in the 4^{th} row, 11^{th} column in the fourth array.

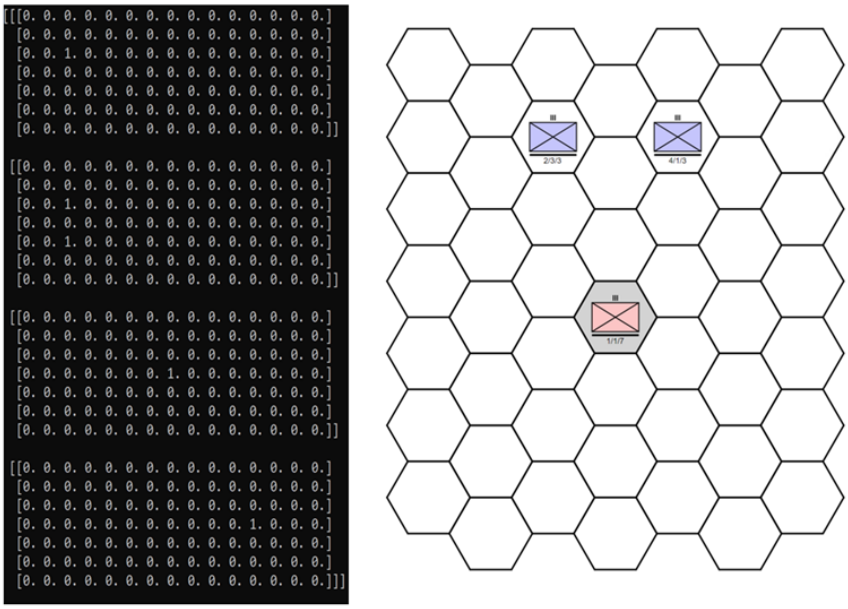

Figure 46. Space Representation with Four Inputs (Mover, Blue Units, Red Units, Terrain).

Unlike any previous scenario where an agent only had to learn a single behavior, the agent now has five different behaviors to learn during its training regimen with each map having its own corresponding behaviors. The performance values needed to achieve optimal performance on each map are summarized in Table 16. Due to the randomness of the shootback AI causing iterations to end with different blue strength values, several maps will have multiple optimal performance values.

Table 16. Changing Urban Terrain Location Optimal Performance Values.

Map	Optimal Blue Strength	Optimal Red Strength	Optimal Total Discounted Reward	Optimal Score
1	100/ 125	0	212.8/ 388/ 436	44/ 81.5/ 129.5
2	150/ 125	0	436/ 484	336/ 334
3	200	100	480	480
4	150	0	340	240
5	150/ 125	0	436/ 484	336/ 334

For Map 1, both blue units must converge on the red unit, destroy it, then have one blue unit occupy the urban terrain hexagon. The blue units will require three attacks on the red unit before destroying it due to the defensive advantage given by the urban terrain. In Maps 2 and 5, the behavior remains consistent with the Urban Terrain scenario when it was valued at 20 points. If the blue unit closest to the urban terrain hexagon is attacked first after they converge on the red unit, it will move into the urban terrain hexagon, leaving the other blue unit with full health to independently destroy the red unit. If the unit opposite to the urban terrain hexagon is attacked first, then both units will stay to destroy the red unit before one occupies the urban terrain hexagon. In Map 3, one blue unit must occupy the urban terrain hexagon and remain stationary for the entirety of the iteration. The other blue unit should do nothing, leaving the red unit completely untouched. Lastly, in Map 4, both blue units need to attack and destroy the red unit simultaneously, then have one blue unit quickly occupy the urban terrain hexagon.

2. Results

Five separate agents were trained for 3,000,000 training steps. After their training, each agent was evaluated based on their ability to achieve optimal performance on all five maps. Only the agent's performance after 3,000,000 training steps was used. As a consequence of the multiple optimal performance values for each map, the method for displaying the agent's results needed to be simplified from the previous scenarios. The percentage of optimal performance repetitions each agent performed during its evaluation was recorded.

Given multiple maps, each with different urban terrain hexagon locations, the agents were able to learn the optimal behaviors for most of the configurations. Using the additional input feature that denotes the terrain location, the agents properly adjusted their behavior to receive the highest discounted reward for each location. As shown in Table 17, for Maps 1–4, all agents displayed optimal performance for all 100 repetitions during their evaluation. Map 5, however, was deemed challenging for most agents as only Agent 2 was able to learn the optimal behaviors. All the other agents failed to recognize the discounted reward benefit of destroying the red unit, and their only tactic entailed solely occupying

the city. Map 2 and Map 5 were the most complex configurations in the scenario as they required the most robust behaviors for optimal performance. Although successful for Map 2, the agents were not able to learn the robust behaviors during the exploration phase.

Table 17. Changing Urban Terrain Location Percentage of Optimal Performance Evaluation at 3,000,000 Training Steps.

	Map 1	Map 2	Map 3	Map 4	Map 5
Agent 1	100.0%	100.0%	100.0%	100.0%	0.0%
Agent 2	100.0%	100.0%	100.0%	100.0%	100.0%
Agent 3	100.0%	100.0%	100.0%	100.0%	0.0%
Agent 4	100.0%	100.0%	100.0%	100.0%	0.0%
Agent 5	100.0%	100.0%	100.0%	100.0%	0.0%

In Map 1, each of the agents used the simultaneous massing technique displayed in each of the previous scenarios, where both blue units entered into the red unit's combat range during the same turn (see Figure 47). Due to the defensive advantage awarded to the red unit for occupying the urban terrain hexagon, the trained agent needed three attacks to finally destroy the red unit. Immediately following the attack, one blue unit occupied the urban terrain hexagon and remain stationary for the remainder of the repetition. As a result of the red unit's occupation of the urban terrain hexagon for 7 or 8 turns before it is destroyed, the resulting score for all runs (44, 81.5, or 129.5) was lower than the total discounted reward (212.8, 388, or 436).

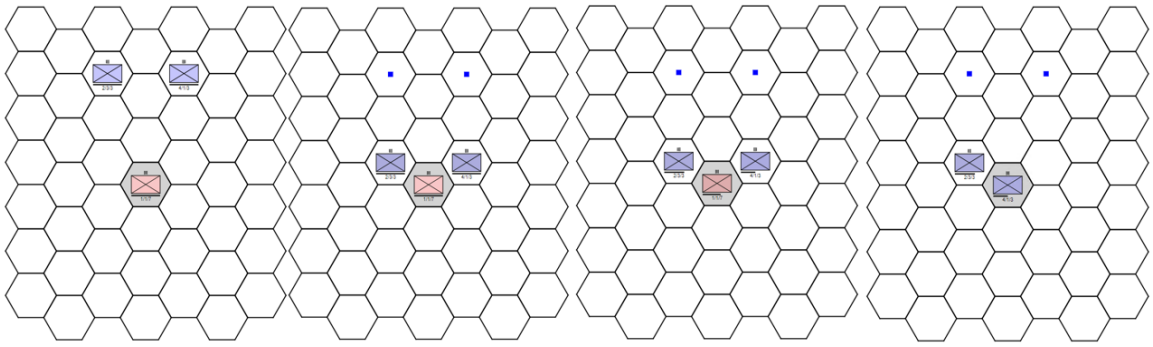

Figure 47. Changing Urban Terrain Location Map 1 Optimal Performance Demonstration.

In Map 2, the agents identified they would receive the highest total discounted reward if the red unit is attacked and destroyed before occupying the urban terrain hexagon (see Figure 48). In their evaluation, the agent's behaviors were slightly different depending on which blue unit was attacked first. As described in the scenario description, if the right blue unit was attacked first, it will immediately occupy the urban terrain hexagon while the left blue unit destroys the red unit independently. However, if the left unit is engaged first, it no longer has the advantage, so the blue unit must stay for the attack until the red unit is killed before occupying the urban terrain hexagon. Depending on the tactics displayed, the trained blue received a final score of 336 or 334 and a total discounted reward of 436 or 484 for each repetition.

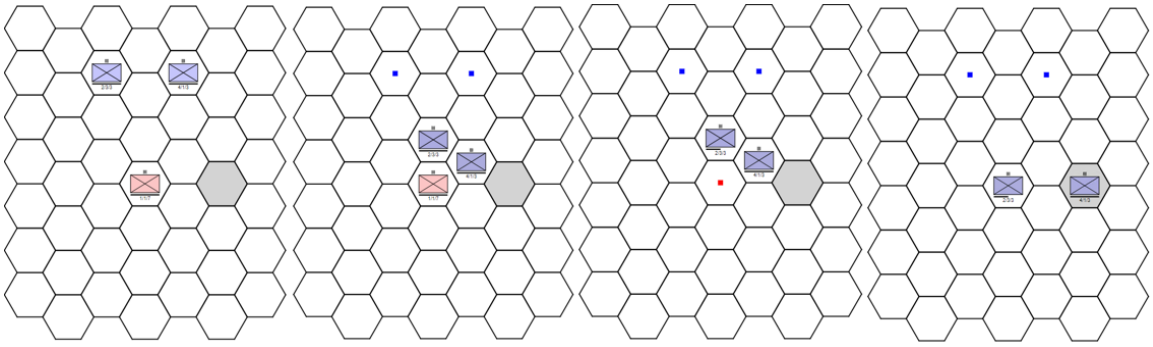

Figure 48. Changing Urban Terrain Location Map 2 Optimal Performance Demonstration.

Map 3 was the simplest of all the urban terrain hexagon configurations. Depicted in Figure 49, all the trained agents quickly identified they would receive the highest reward for solely occupying the urban terrain hexagon for the entirety of each repetition. Each agent earned 480 points for their final score and total discounted reward.

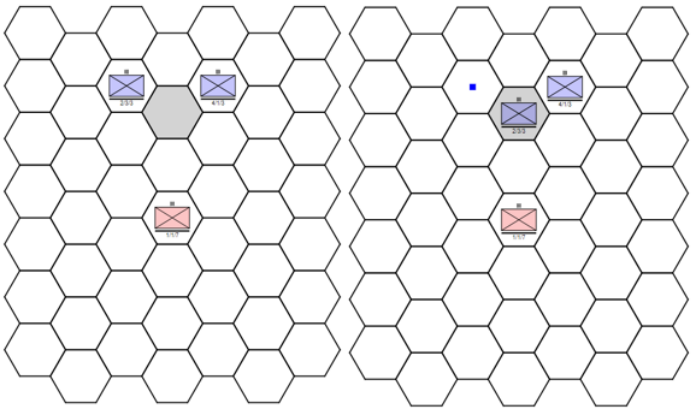

Figure 49. Changing Urban Terrain Location Map 3 Optimal Performance Demonstration.

The behaviors displayed by the trained agents during the evaluation for Map 4, as demonstrated in Figure 50, are relatable to the behaviors seen in Map 1. The agents first started by converging on the red unit to destroy it. The blue units only required two attacks during their engagement due to the red unit no longer having the defensive advantage. Once destroyed, the blue takes the fastest route to occupy the urban terrain hexagon on the 13th turn. For all repetitions during their evaluation, each agent earned a score of 240 and a total discounted reward of 340.

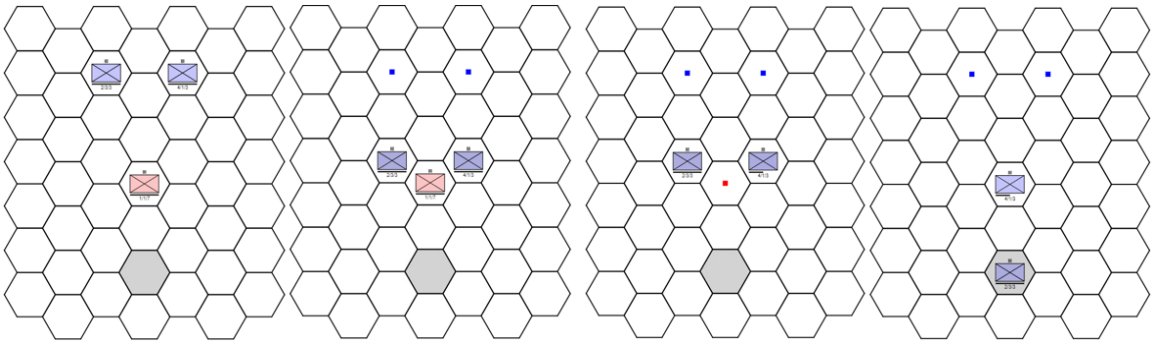

Figure 50. Changing Urban Terrain Location Map 4 Optimal Performance Demonstration.

The behaviors required to achieve optimal performance in Map 5 were the same as in Map 2, except it needed to be the left blue unit deciding when to occupy the urban terrain

hexagon vise the right blue unit. Although all trained agents learned the optimal behaviors for Map 2, only one agent learned the optimal behaviors needed for Map 5. As shown in Figure 51, four agents went straight to occupying the urban terrain hexagon instead of initially destroying the red unit. Not learning the optimal behaviors caused the agents to earned a total discount reward of 384 points instead of the 436 or 484 points they could have earned by using the optimal behaviors.

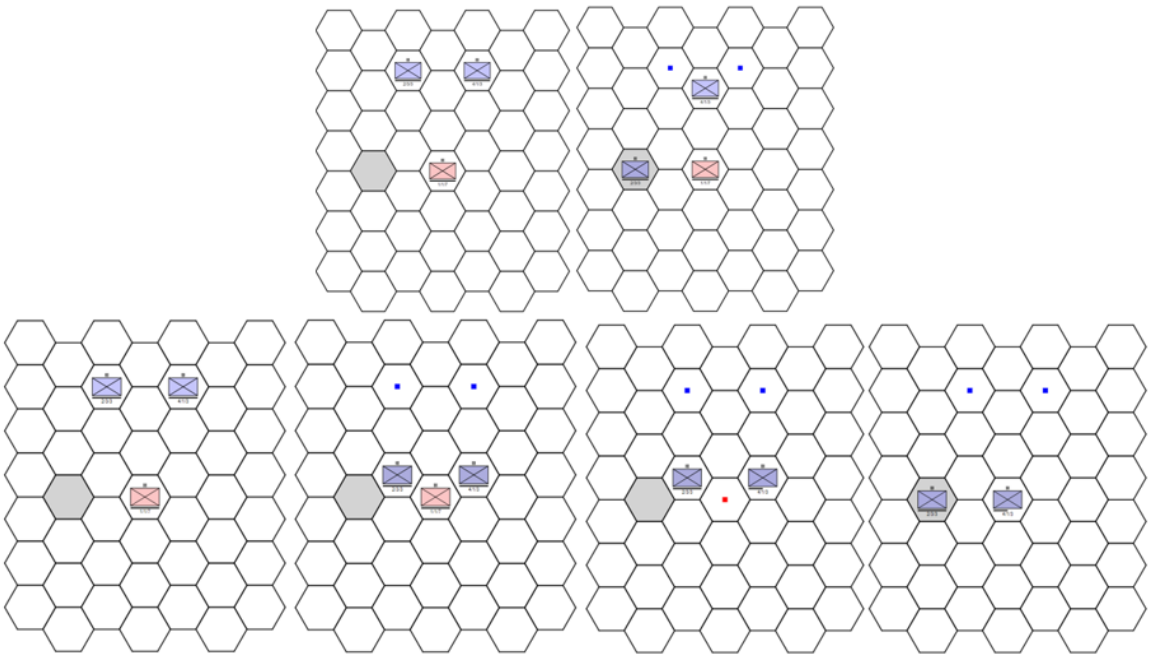

Figure 51. Changing Urban Terrain Location Map 5 Non-Optimal (top) and Optimal (bottom) Performance Demonstration.

G. MULTI-AGENT TRAINING

1. Description

Unlike in the other scenarios, the purpose of the Multi-Agent Training scenario is to train agents for both the blue and red force that are capable of learning behaviors to combat each other's tactics. The training scenario involves a two-versus-one configuration where two blue units attack a single red unit. To set up a battlefield where both forces have the ability to achieve success, additional rough and urban hexagons were added to offer

cover to an occupying unit. An impassable barrier with two passages was additionally built-in to the battlefield to separate the blue setup zone from the red setup zone (see Figure 52). The training regimen for the scenario starts by training a blue agent against a shootback red AI. The expected behavior of trained blue agent is to engage the red unit by moving one of the blue units into the rough terrain field north of the red unit. Using the terrain's defensive advantage after occupation, the train blue agent should destroy the red unit and be left with 187.5 strength points.

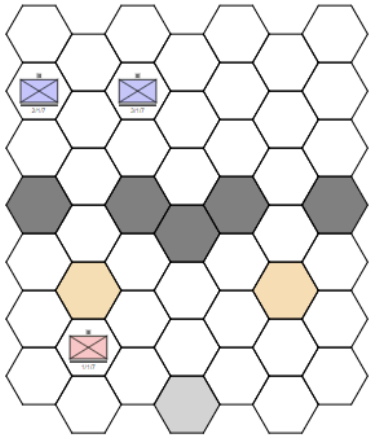

Figure 52. Multi-Agent Scenario.

Following a successful training of the blue agent, its behaviors will now be used as the opponent to train a red agent. For the red agent to achieve optimal performance, it must learn to move its unit north into the rough terrain field during its initial moves. Holding that position, the red unit will now have the advantage needed to destroy both blue units during their attack.

For the scenario, the default reward function and network structure was used. For the training of the red agent, the score values were adapted in a way that a loss on the blue side generated a score value of 2 while a loss on the red side generated a score value of -1.

2. Results

In a first attempt, a blue agent successfully learned to defeat the red force by moving one unit into the rough hexagon north of the red unit. However, when attempting to use the blue agent to train the red agent, the blue agent could no longer show any reasonable behaviors. Introducing a red force that could now take actions other than what was directed by the shootback AI created states that were now unrecognizable to the blue agent when fed into the neural network as inputs. This resulted in having the blue agent take random actions for the entirety of the red agent's training.

Another blue agent was trained against a red force using the random move AI in an attempt to correct the previous blue agent's behavior. As described earlier, the random move AI allows the red force to randomly move into an adjacent hexagon 40 percent of the time unless a blue unit is within combat range. Following the training, the blue agent exhibited three different behaviors depending on the current position of the red unit. The blue agent either proceeded towards the city, moved onto the rough hexagon, or did not attack at all. As a general pattern, the blue agent moved towards the city, whenever the red unit was close to the lower left of the map (see Figure 53). If the red unit was in its initial setup position, then the blue unit would opt to destroy it first before moving into the city.

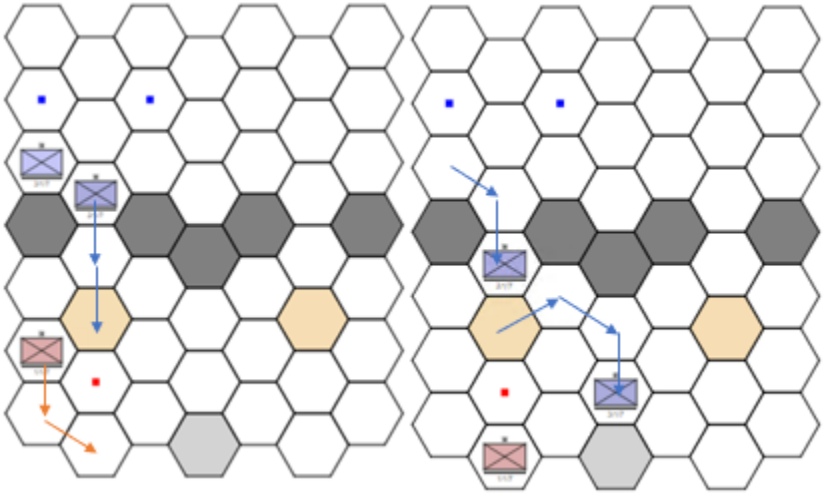

Figure 53. Movement of Blue and Red Agents.

The trained blue agent was then used as the opponent to train three red agents for 5,000,000 training steps each. The first agent showed a behavior where it would first move towards the upper right direction before turning and start moving towards the city. Since one of the blue agent's behaviors was to occupy the urban terrain hexagon, the red agent decided to meet the blue unit there to attack the blue unit. Unfortunately, the blue agent was still unable to recognize an increased dynamic state from its training, so it typically did not shoot back against the red unit. Although the blue agent had the defensive advantage in the urban terrain and outnumbered the red agent, this battle was typically won by the red unit (see Figure 54). The other two trained red agents showed a behavior in which they would move to the upper right direction and position on the right rough hexagon. However, these two agents did not proceed with an attack on the blue unit during any of the repetitions.

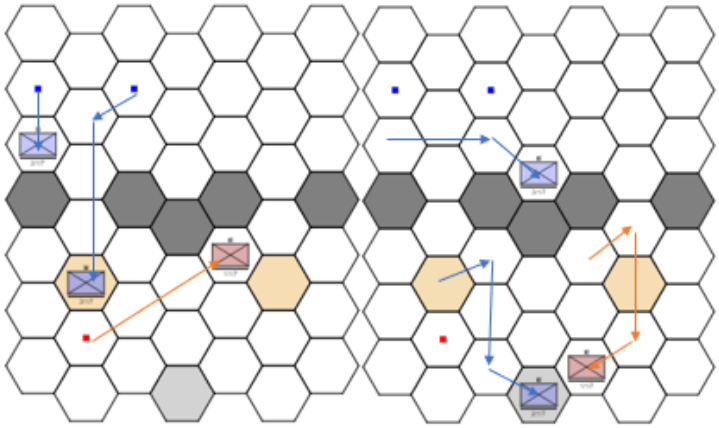

Figure 54. Battle Development of Blue Versus Red Agent.

To confront the shortcomings of an unstable blue behavior on a map with limited blue and red set up hexagons, the amount of possible set up zones was expanded in the next step (Figure 55). Keeping the neural network structure, reward function, and random AI constant, another blue agent was trained for 10,000,000 training steps. The resulting blue agent's behavior highly depended on the position of the red unit. The most common behavior when the red unit was on the eastern side of the map involved the trained blue agent moving both units to the southern portion of the map to occupy the urban terrain

hexagon. The scenario typically ended with one blue unit positioned in the urban terrain hexagon and one blue unit positioned in the left rough hexagon (see Figure 55).

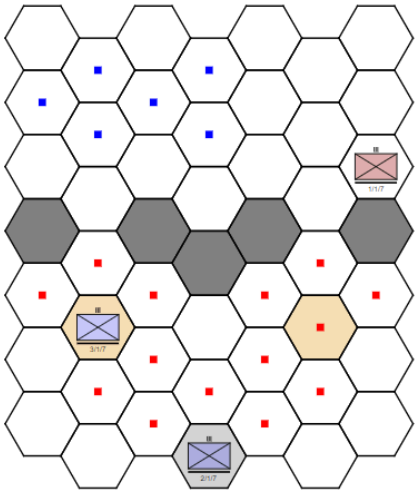

Figure 55. Standard Behavior of Blue Player in Complex Setup.

However, when the random AI positioned the red unit adjacent to the rough hexagon, the blue agent attacked the red unit from the rough hexagon. After the attack of the red unit, the blue agent moved the then damaged unit away and replaced it with the other full-strength blue unit positioned in the back (see Figure 56). The trained blue agent would then destroy the red unit and conquer the urban hexagon.

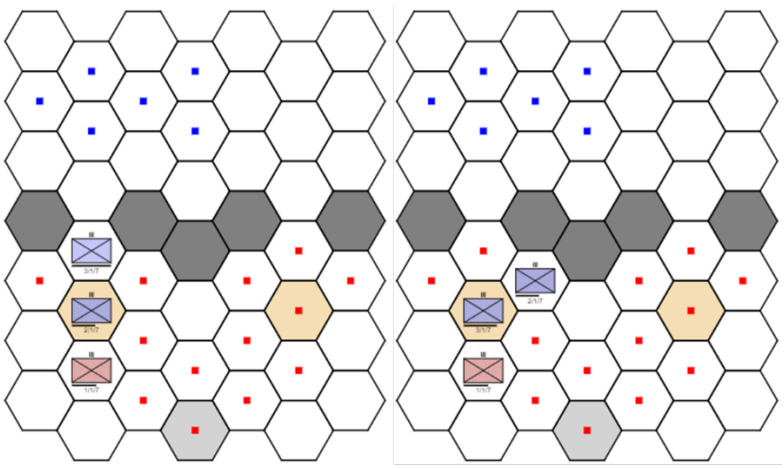

Figure 56. Blue Behavior when Red Unit is Adjacent to Rough Hexagon.

For occurrences when the passageway from the northern part of the map to the southern part was blocked, the blue agent would use its first unit to damage the red unit as much as possible. After the destruction of his first unit by the red unit, it would then use its remaining strength to destroy the red unit and conquer the city (see Figure 57).

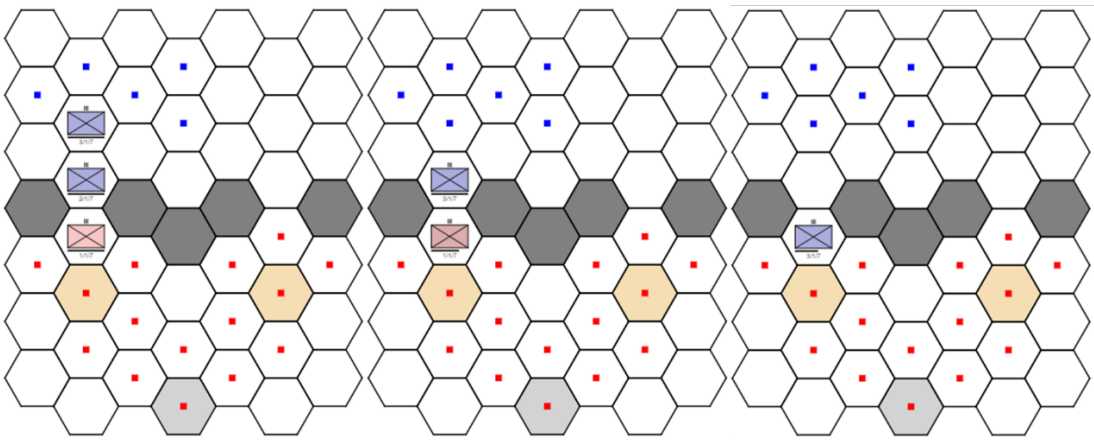

Figure 57. Blue Behavior when Red locks Passage.

Using the newly trained blue agents as opponents, two red agents were trained for 2,000,000 training steps. However, both agents always moved southwards no matter which behavior was shown by the blue agent. A reasonable explanation for this behavior could

not be found other than human error while setting up the code for multi-Agent Learning. Unfortunately, due to time constraints, an additional red agent could not be trained to correct these issues.

H. LARGE-SCALE SCENARIO

1. Description

In this scenario, a blue force of twelve units featuring armor, artillery, and mechanized infantry unit types competes against a six-unit red force composition that included infantry, artillery, and armor unit types (see Figure 58). The setup positions for each unit remain fixed for the entirety of the scenario. As described at the beginning of the chapter, the default neural network structure and reward function were used to train the blue agent. With an increased terrain size and number of units compared to any of the other scenarios, the number of turns in a game was increased from 20 to 30. Due to the high complexity of the scenario, an optimal score and reward could not be calculated as the number of possible actions was too high. However, the theoretical score and reward could be inferred based on the assumption that it would be possible for the blue agent to destroy all red units without having any blue unit killed and conquer the urban hexagon in a minimal amount of time. However, these values would not give any further insight when comparing different agents.

In total, three agents will be trained, with each using different input layers to represent the state of the scenario. One agent will use the default input layers as described in Chapter III. The second agent will have four additional input layers to represent all of the unit types in the scenario for a total of seven layers. The third agent will be the same as the second agent, except an additional layer will be added to represent the rough terrain locations for eight inputs total. Each agent will be trained for 3,000,000 training steps, and the resulting behaviors will be analyzed.

In a separate analysis to compare the performance of an MLP neural network to a CNN in a large-scale scenario, two additional agents will be trained for 10,000,000 training steps. Each agent will have four input layers, including three default layers and an

additional one representing the artillery units. Following their training, the performance of the two agents will be assessed.

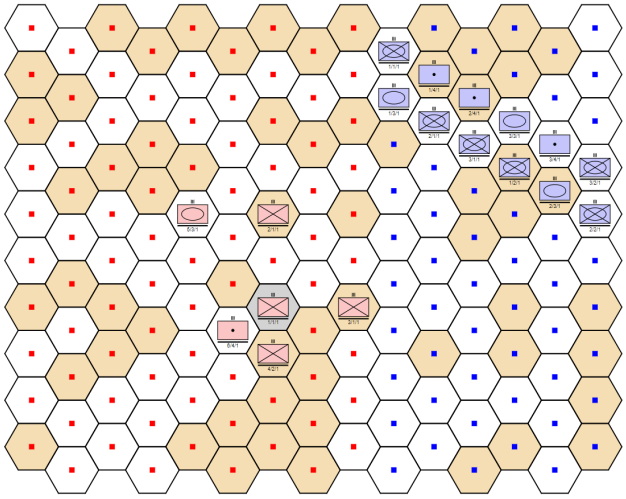

Figure 58. Large-Scale Scenario Setup.

2. Results

As shown in Figure 59, the structure and amount of input layers given to the neural network do not seem to make a difference for the agent's performance. The achieved total discounted rewards each show a similar pattern for all three agents during the training process, where a maximum of nearly 400 points is reached after approximately 2,000,000 training steps. While the second agent with seven input layers showed some instability during the training process, the other two agents seemed to remain stable. For the other two agents with only the added artillery unit layer used for the CNN and MLP network comparison, the training was expanded up to 10,000,000 training steps. However, even with that additional training steps, the performance did not improve further.

Figure 59. Reward Compared between Different Input Layer Variants.

When comparing the trained CNN agent with the MLP agent, both show similar promising behaviors (see Figure 60). The CNN agent starts by initially moving its units southwest in a large formation. Once the first contact is made with a red unit on the northwest side, the south units shift their movement and start heading west. In a combined effort, the trained agent continues moving westward to attack the remaining red units further while consistently keeping the artillery units in the rear to take advantage of its more extended range. The scenario typically ends due to time constraints, but the agent was able to destroy most of the red units and occupy the urban terrain hexagon.

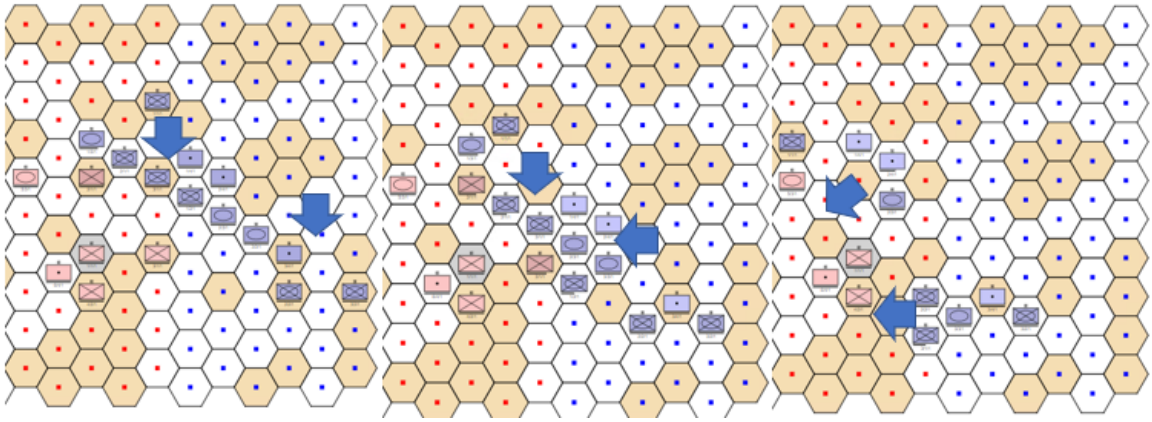

Figure 60. Behavior of a CNN Agent after 30,000,000 Training Steps.

The MLP agent, however, used a slightly different approach. At the start of the scenario, the agent takes advantage of the increased speed for mechanized units when traveling over clear terrain. Three blue units quickly maneuver to destroy the most eastern red infantry unit, which allows for a clear passageway to southern red units. The agent then splits its forces to attack the two red units at the north of the map simultaneously as the three units at the south of the map using similar tactics shown by the CNN agent. The faster movements allow the MLP agent to conquer the urban terrain hexagon and destroy all the red units within the time constraints, as shown in Figure 61.

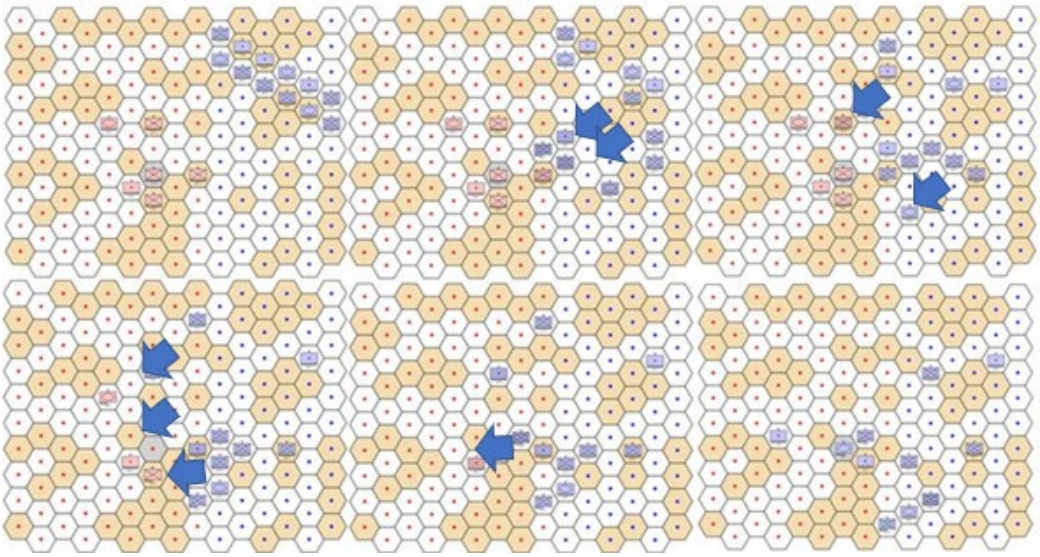

Figure 61. Behavior of MLP Agent after 30,000,000 Training Steps.

As expected, based on the more effective behaviors during initial combat, the MLP agent achieved an average higher total discounted reward and score than the CNN agent (see Figure 62). Still, it should be reminded that only one agent was trained for each neural network, so there is nowhere near enough evidence to decide which structure is a better fit for training a combat agent in a larger scenario. Both structures, however, did show the capability to learn coordinating behaviors for scenarios with far more complexity than previously encountered.

Figure 62. Score Progression and Adjusted Reward of MLP and CNN Agent.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS, RECOMMENDATIONS, AND FUTURE WORK

The objectives of this thesis focused on determining whether a CNN can be used to train agents capable of learning the optimal behaviors achieved by Boron [6], as well as in scenarios of increased complexity. Furthermore, this thesis explored whether a multi-agent training regimen can be utilized in the domain of military constructive simulations. In total, eight different scenarios were created in the AI training environment, Atlatl, all varying in size, number of units, unit and terrain types, or scripted red force behaviors. Overall, the objectives were accomplished as the agents demonstrated robust attacking behaviors in each of the scenarios, and a basic understanding of the applications Atlatl can be applied to multi-agent training was accomplished. In this chapter, the principal conclusions of the scenarios will be summarized, recommendations for improvement will be mentioned, and suggestions for future work will be offered.

A. CONCLUSIONS

In this sub-chapter, the conclusions will be provided for each scenario category. The sub-chapter starts with the finding when replicating the scenarios described by Boron [6].

1. Replication of Boron's Scenarios

Although the scenarios were modeled slightly differently due to the differences in the AI training environments, the agents using a CNN as the neural network structure were able to achieve optimal performance for all three simple scenarios described in Boron's [6] research. Each agent understood the importance of coordinating each of the blue units' actions to ensure simultaneously massing of forces when conducting an attack. In most situations, two blue units coordinated their movements to enter hexagons adjacent to the red unit on the same turn. Using these tactics guaranteed the agents received minimal casualties and allowed them to maximize the total discounted reward in each of the scenarios. While the scenarios with fixed starting positions were too simple enough to require the spatial invariance advantages a CNN can provide, they did confirm that a CNN

structure is an effective tool when using RL to train combat agents in constructive simulations.

2. Increasing the complexity of Boron's Scenarios

As scenarios consisting of multiple starting positions and formations were introduced, the spatial invariance advantages of CNN were leveraged more frequently than previously described in the more straightforward scenarios. Instead of everything remaining constant during the agents' training regimen, the units were positioned randomly throughout the map, depending on the scenario. The agents were now required to learn multiple behaviors in a single scenario to continuously defeat the red force. Even with the increased complexity, most agents could achieve optimal performance for each scenario in the same amount of training steps required to learn the optimal performance when the configurations were fixed. The CNN's ability to recognize similar features or patterns in the unit positions allowed the agents to deduce the required behavior for optimal performance quickly. Additionally, the convolutional layers allowed an agent to learn the behaviors needed to defeat a retreating enemy. Although the opponent's behaviors was simple and only near-optimal performance was achieved, an agent capable of learning how to defeat a moving enemy shows the potential of using CNN in scenarios of much higher complexity.

3. Spatial Invariance

Although the different state representations for units located in odd columns compared to even columns caused the agent to exhibit spatial invariant behavior for only six out of fourteen possible configurations, that issue is specific to the Atlatl training environment and should not discourage future researchers from using similar approaches in their experimentations. While using arrays with the double coordinate system helped ensure a fixed convolutional kernel can see the same set of neighbors regardless of where it is centered, there remained several differences in the representation of units located in adjacent rows. Instead of using a fixed convolutional kernel, a possible solution to address the state representation issue is to incorporate a hexagonal convolution tool such as HexagDLy [42]. HexagDLy is open-source software that is specifically designed for

convolutional operations on hexagonal state-spaces, where it translates the hexagonal information onto a 2-dimensional grid that can be processed as inputs for frameworks such as PyTorch. Overall, developing fully spatially invariant agents capable of performing the same optimal behaviors no matter where it is located on the map has the potential to decrease the time required for training significantly and can allow for more robust agents.

4. Scenarios with Urban Terrain Features

Learning the optimal behaviors in scenarios featuring an urban terrain hexagon did not seem to be a challenging task for each of the agents. The agents quickly identified the terrain feature and exhibited different behaviors depending on its value or location. Simply adjusting the value of the urban terrain value to represent cities of varying importance forced the agents to learn three distinct behaviors, with each maximizing the total discounted reward for its configuration. This straightforward implementation can allow for more dynamic agents that exhibit multiple behaviors in a single scenario within a constructive simulation. By introducing an urban terrain hexagon that changes position during the training regimen, an additional input layer for the neural network was required. Even with the added layer, though, the neural network effectively processed the information needed to achieve optimal performance or near-optimal performance on all five urban terrain locations. A single agent trained only once can now perform numerous distinct actions for multiple configurations inside of a scenario through this training process.

5. Multi-Agent Learning

From the results, it is clear that using a multi-agent learning approach in Atlatl requires some amount of variability during the training process. Before OpenAI's AlphaStar even entered the competition phase in StarCraft II, where multi-agent training occurred, it had spent days using a combination of supervised learning and RL [5]. Studying over 900,000 replays from the top 22 percent of players provided AlphaStar the variability needed to understand the regularly occurring situations in the game. Not having the resources or library of replays for the agents to study in Atlatl, the agents were left relying on the behavioral scripts, which were visibly not enough. Without this variability,

the trained agents rarely recognized a familiar state and resorted to unexplainable behaviors as their default. While the resulting agent behaviors are not as strong as this research hoped for, a better understanding of the training process required has been accomplished.

6. Large-Scale Scenario

Based on the Large Scale Scenario, it appears that in a static setup and map design, additional input layers do not bring additional benefits in the sense of decreased learning time or better results. Due to the consistent results shown when varying the number of input layers, the additional amount of information did not seem to benefit the agents in any way. The simplicity of the scenario allowed for the training steps to be enough for agents to teach themselves the behaviors that provided the largest discounted reward. However, additional input layers are necessary for future work if more complex or moving terrain features are introduced, as shown in the Changing Urban Terrain Location Scenario.

While the resulting CNN and MLP structure trained agents showed interesting tactical behaviors during their evaluation, there was no way to validate if their actions were optimal. However, a more appropriate and promising determination is that the methodology in this thesis has the potential to tackle much more extensive and complex scenarios in the future.

B. RECOMMENDATIONS AND FUTURE WORK

Due to the exploratory nature of this thesis, hyperparameter optimization, as it usually takes place with comparable papers, did not occur. The solutions found are therefore not optimal in terms of the training time required. Particularly when the scenarios become more complex regarding the allowed starting positions during the setup phase for the units or include more dynamic scripted red force behaviors, the training time in this thesis has increased drastically. Future works should therefore start with a hyperparameter optimization before continuing with more complex scenarios. Another possibility for optimizing the required training time is the use of hexagonal filters within the CNN structure. The infrastructure used in the context of this thesis is designed to recognize patterns in images containing a rectangular pixel scheme. To compensate for the hexagonal structure, a double coordinate system with the stretching of the y-axis was used to ensure

a fixed convolutional kernel can see the same set of neighbors regardless of where it is centered when using 2-dimensional arrays. However, filter shapes that are specifically tailored to hexagonal fields could be more effective for future work.

The used Atlatl-Framework uses a scoring system in which a deduction of a single blue strength value is worth -2 points while a deduction of a single red strength value is worth 1 point. These values were introduced arbitrarily to the Atlatl framework. However, matches that typically ended with scores of negative or zero values were not conducive to training an offensive agent. The trained agents would rather avoid the red units altogether than learn the optimal behavior with the negative rewards from taking damage outweighing the positive rewards achieved by attacking. To encourage attacking behaviors from the agents during training, all negative rewards were then ignored, and the positive rewards were discounted proportionally to the number of blue units still alive.

For the multi-agent approach, this thesis showed that a trained agent is sensitive to a change in the opponent's behavior whenever it encounters behaviors never experienced before. The AI scripted behaviors used within this thesis to train each agent initially were simply structured and could perform only a few actions. When competition between the two agents began, the drastic increase of behaviors exhibited by the opposing force was too foreign for the agent to respond appropriately. Improving and increasing the complexity of scripted enemy behaviors for multi-agent training in the future could potentially produce better insights into the usage of multi-agent training for military applications. Another promising way to deal with the stability of an agent's behavior when increasing the complexity of a scenario could be the implementation of a league-based training system. In combination with a more dynamic enemy, it should be investigated how the combination of deep-RL with other ML techniques can be used to produce more stable behaviors with fewer training steps. Sun et al. [7]. incorporated an approach that combined deep-RL with PKQ-Learning to develop an agent capable of exhibiting many different behaviors. The utilization of these or similar technologies might allow more agents capable of performing optimal behavior in more complex scenarios with fewer training steps.

In general, this thesis and comparable theses, done at the MOVES Institute before, are theoretical in nature. Future research should investigate how deep-RL techniques can

be integrated into existing DOD programs of record constructive simulation systems such as MTWS, Combat XXI, or OneSAF for a more practical benefit. Part of the research should focus on developing an API capable of integrating deep-RL tools like StableBaselines3 into these simulations, and it should be structured according to architecture and interfaces.

C. SUMMARY

This thesis confirmed the applicability of using deep-RL techniques to develop robust artificial agents capable of achieving optimal performance in scenarios featuring multiple unit and terrain types. Additionally, the foundation for multi-agent learning and fully spatially invariant agents was established. While the work presented strengthens the promising efforts of these tools in military research, all of the results were accomplished using heavily abstracted scenarios in an AI training environment specially designed to incorporate external RL applications into its framework. Additional research should look to discover an approach to apply similar methods to an existing DOD program of record constructive simulation. Having this capability to act as the red force in existing constructive simulations could save human resources and allow commanders to test current tactics, highlight any vulnerabilities in their current plan, and validate their scheme of maneuver before conducting a single live rehearsal or execution.

APPENDIX. DATA

A. TWO-VERSUS-ONE

Fixed-Positions

Training Steps	Agent 1		Agent 2		Agent 3		Agent 4		Agent 5	
	%-perfect game	Score	%-perfect game	Score	%-perfect game	Score	%-perfect game	Score	%-perfect game	Score
100000	1	0	0	-107.5	0.8	-52.5	0.9	-20	0.9	-5
200000	0.9	-10	0	-77.5	1	0	1	0	1	0
300000	1	0	0	-90	0.9	-17.5	0.3	-57.5	0.9	-10
400000	0.7	-72.5	0	-87.5	1	0	1	0	0.8	-35
500000	1	0	0	-90	1	-160	0.6	-35	1	0
600000	0	-175	0	-87.5	1	0	1	0	0.9	-20
700000	0	-175	0	-87.5	0	-132.5	1	0	1	0
800000	0	-205	0	-85	1	0	1	0	1	0
900000	0	-142.5	0	-92.5	1	0	0.8	-22.5	1	0
1000000	0	-145	0	-90	0.9	-15	1	0	1	0
1100000	0	-170	0	-85	1	0	1	0	1	0
1200000	0	-142.5	0	-97.5	1	0	1	0	0.9	-20
1300000	0	-145	0	-82.5	1	0	0.9	-7.5	1	0
1400000	0	-117.5	0	-82.5	1	0	0.7	-27.5	1	0
1500000	0	-172.5	0	-92.5	1	0	1	0	1	0
1600000	0	-145	0	-85	1	0	1	0	1	0
1700000	0	-147.5	0	-90	1	0	1	0	0	-122.5
1800000	0	-132.5	0	-87.5	1	0	0.9	-10	1	0
1900000	0	-150	0	-120	1	0	1	0	1	0
2000000	0	-150	0	-100	1	0	1	0	1	0

Fixed-Formation

Training Steps	Agent 1		Agent 2		Agent 3		Agent 4		Agent 5	
	%-perfect game	Score	%-perfect game	Score	%-perfect game	Score	%-perfect game	Score	%-perfect game	Score
100000	0.1	-157.5	0	-165	0	-175	0	-175	0	-175
200000	0.1	-122.5	0	-147.5	0.1	-170	0	-155	0.2	-140
300000	0.1	-177.5	0.1	-107.5	0	-187.5	0	-182.5	0	-175
400000	0.1	-147.5	0.1	-162.5	0	-185	0	-152.5	0	-175
500000	0	-165	0.4	-50	0	-175	0	-147.5	0	-175
600000	0	-182.5	0.6	-55	0.2	-127.5	0.1	-145	0	-155
700000	0.1	-150	0.2	-85	0.1	-157.5	0.1	-177.5	0	-162.5
800000	0.2	-132.5	0.4	-70	0.1	-177.5	0.2	-140	0.1	-147.5
900000	0.1	-140	0.1	-85	0.1	-157.5	0	-152.5	0.1	-150
1000000	0.1	-127.5	0.3	-57.5	0	-175	0	-162.5	0.4	-105
1100000	0.1	-147.5	0.5	-37.5	0.1	-157.5	0.3	-135	0.3	-112.5
1200000	0.2	-112.5	0.3	-55	0.1	-157.5	0.2	-112.5	0.1	-140
1300000	0.8	-35	0.4	-62.5	0.2	-140	0.3	-95	0.3	-92.5
1400000	0.2	-132.5	0.2	-122.5	0.1	-157.5	0.5	-75	0.3	-90
1500000	0.4	-90	0.5	-105	0.1	-150	0.1	-142.5	0.6	-60
1600000	0.7	-32.5	0.5	-37.5	0.1	-177.5	0.4	-65	0.5	-107.5
1700000	0.7	-35	0.4	-42.5	0.3	-122.5	0.5	-65	0.4	-77.5
1800000	0.8	-35	0.6	-32.5	0	-175	0.4	-62.5	0.6	-50
1900000	1	0	0.3	-95	0.3	-122.5	0.3	-72.5	0.2	-140
2000000	0.7	-72.5	0.3	-72.5	0.4	-105	0.4	-85	0.6	-55
2100000	0.9	-17.5	0.2	-92.5	0.3	-122.5	0.5	-55	0.6	-45
2200000	1	0	0.4	-75	0.2	-140	0.4	-42.5	0.6	-47.5
2300000	1	0	0.7	-25	0.3	-122.5	0.3	-65	0.4	-67.5
2400000	0.6	-70	0.7	-22.5	0.2	-140	0.3	-72.5	0.7	-20
2500000	0.9	-17.5	0.2	-62.5	0.3	-122.5	0.6	-52.5	0.4	-57.5
2600000	0.8	-35	0.3	-47.5	0.8	-35	0.3	-62.5	0.6	-55
2700000	0.9	-17.5	0.5	-100	0.8	-35	0.3	-62.5	0.8	-15
2800000	0.7	-52.5	0.7	-22.5	0.2	-140	0.7	-35	0.6	-45
2900000	0.9	-17.5	0.5	-45	0.4	-105	0.7	-22.5	0.4	-50
3000000	1	0	0.3	-60	0.1	-157.5	0.4	-100	0.5	-37.5
3100000	0.9	-17.5	0.4	-62.5	0.2	-140	0.5	-65	0.8	-15
3200000	0.9	-17.5	0.5	-42.5	0.4	-105	0.4	-72.5	0.6	-35
3300000	0.8	-35	0.6	-32.5	0.3	-122.5	0.3	-67.5	0.5	-52.5
3400000	0.7	-52.5	0.4	-50	0.2	-140	0.3	-82.5	0.6	-32.5
3500000	0.9	-17.5	0.4	-50	0.5	-87.5	0.5	-47.5	0.6	-32.5
3600000	0.9	-17.5	0.3	-57.5	0.6	-107.5	0.5	-42.5	0.7	-15
3700000	0.8	-25	0.3	-52.5	0.6	-70	0.4	-60	0.7	-25
3800000	0.7	-52.5	0.7	-27.5	0.5	-80	0.6	-42.5	0.6	-35
3900000	0.9	-17.5	0.8	-17.5	0.4	-97.5	0.7	-32.5	0.6	-32.5
4000000	1	0	0.5	-42.5	0.3	-82.5	0.4	-62.5	0.8	-27.5
4100000	1	0	0.5	-72.5	0.4	-97.5	0.4	-57.5	0.8	-25
4200000	1	0	0.9	-5	0.4	-105	0.6	-42.5	0.6	-32.5
4300000	0.8	-35	0.7	-20	0.7	-52.5	0.5	-40	0.8	-15
4400000	1	0	0.7	-57.5	0.6	-47.5	0.6	-40	0.6	-30
4500000	0.9	-17.5	0.4	-70	0.4	-67.5	0.4	-55	0.5	-35
4600000	0.6	-70	0.8	-15	0.4	-57.5	0.5	-40	0.8	-17.5
4700000	0.8	-35	0.8	-15	0.7	-42.5	0.5	-50	0.6	-37.5
4800000	0.8	-35	0.7	-22.5	0.7	-47.5	0.5	-65	0.6	-32.5
4900000	0.8	-35	0.9	-10	0.7	-27.5	0.4	-45	0.7	-25
5000000	1	0	0.8	-15	0.8	-20	0.7	-17.5	0.7	-25

Random

Training Steps	Agent 1		Agent 2		Agent 3		Agent 4		Agent 5	
	% perfect Games	Score	% perfect Games	Score	% perfect Games	Score	% perfect Games	Score	% perfect Games	Score
100000	0.6	-92.5	0.4	-160	0.7	-75	0.3	-175	0.3	-155
200000	0.3	-170	0	-197.5	0.4	-137.5	0.2	-155	0.1	-185
300000	0.4	-185	0.4	-175	0.1	-237.5	0	-182.5	0	-272.5
400000	0	-247.5	0.3	-162.5	0.4	-112.5	0	-152.5	0.1	-232.5
500000	0.1	-177.5	0.3	-117.5	0.1	-205	0	-147.5	0	-217.5
600000	0.1	-187.5	0	-195	0.2	-177.5	0.1	-145	0.1	-180
700000	0.1	-185	0.3	-130	0.2	-127.5	0	-177.5	0.5	-82.5
800000	0.2	-127.5	0.2	-155	0.3	-120	0.2	-140	0.1	-212.5
900000	0.4	-107.5	0	-217.5	0.4	-157.5	0	-152.5	0.1	-205
1000000	0.2	-200	0.3	-125	0.4	-142.5	0.4	-162.5	0.4	-120
1100000	0.6	-67.5	0.1	-202.5	0.2	-132.5	0.3	-135	0.3	-132.5
1200000	0.4	-127.5	0.2	-152.5	0.4	-127.5	0.4	-112.5	0.2	-177.5
1300000	0.8	-22.5	0.3	-120	0.4	-77.5	0.3	-95	0.5	-122.5
1400000	0.3	-165	0.3	-87.5	0.3	-90	0.3	-75	0.1	-187.5
1500000	0.5	-85	0.4	-120	0.4	-70	0.1	-142.5	0.5	-72.5
1600000	0.2	-107.5	0.3	-75	0.3	-67.5	0.3	-65	0.1	-160
1700000	0.2	-137.5	0.3	-130	0.5	-72.5	0.4	-65	0.4	-80
1800000	0.3	-130	0.3	-120	0.4	-65	0.6	-62.5	0.5	-120
1900000	0.4	-127.5	0.2	-137.5	0.5	-85	0.3	-72.5	0.6	-80
2000000	0.6	-35	0.2	-107.5	0.3	-130	0.1	-85	0.4	-127.5
2100000	0.5	-65	0.1	-155	0.2	-140	0.3	-55	0.3	-92.5
2200000	0.3	-145	0.7	-70	0.4	-120	0.1	-42.5	0.5	-52.5
2300000	0.3	-137.5	0.3	-135	0.3	-117.5	0.2	-65	0.2	-155
2400000	0.5	-87.5	0.3	-120	0.3	-95	0.2	-72.5	0.6	-62.5
2500000	0.2	-125	0	-117.5	0.4	-95	0.3	-52.5	0.3	-122.5
2600000	0.6	-67.5	0.4	-77.5	0.6	-82.5	0.3	-62.5	0.3	-135
2700000	0.5	-60	0.1	-127.5	0.6	-47.5	0.4	-62.5	0.5	-72.5
2800000	0.4	-70	0.2	-85	0.5	-67.5	0.7	-35	0.6	-45
2900000	0.4	-80	0.3	-85	0.4	-85	0.4	-22.5	0.5	-85
3000000	0.5	-97.5	0.2	-97.5	0.7	-27.5	0	-100	0.3	-117.5
3100000	0.5	-92.5	0.3	-67.5	0.4	-100	0.2	-65	0.4	-87.5
3200000	0.4	-80	0.4	-107.5	0.6	-55	0.5	-72.5	0.3	-120
3300000	0.5	-67.5	0.6	-45	0.4	-82.5	0.6	-67.5	0.4	-122.5
3400000	0.4	-85	0.4	-55	0.5	-95	0.4	-82.5	0.2	-152.5
3500000	0.3	-85	0.4	-100	0.5	-52.5	0.6	-47.5	0.5	-57.5
3600000	0.3	-115	0.2	-97.5	0.6	-75	0.2	-42.5	0.2	-127.5
3700000	0.3	-70	0.7	-67.5	0.4	-85	0.4	-60	0.5	-45
3800000	0.4	-90	0.6	-55	0.5	-80	0.5	-42.5	0.4	-75
3900000	0.2	-92.5	0.5	-57.5	0.5	-55	0.7	-32.5	0.4	-110
4000000	0.6	-67.5	0	-150	0.4	-60	0.3	-62.5	0.2	-147.5
4100000	0.6	-42.5	0.1	-125	0.4	-77.5	0.2	-57.5	0.3	-150
4200000	0.4	-65	0.5	-52.5	0.5	-80	0.4	-42.5	0.4	-85
4300000	0.3	-72.5	0.4	-80	0.6	-47.5	0.4	-40	0.6	-95
4400000	0.3	-97.5	0.6	-35	0.5	-75	0.4	-40	0.5	-55
4500000	0.4	-72.5	0.3	-87.5	0.5	-60	0.3	-55	0.4	-80
4600000	0.3	-90	0.6	-35	0.4	-72.5	0.2	-40	0.6	-57.5
4700000	0.6	-80	0.4	-105	0.7	-47.5	0.6	-50	0.2	-105
4800000	0.3	-110	0.2	-82.5	0.5	-60	0.3	-65	0.3	-67.5
4900000	0.6	-52.5	0.4	-70	0.5	-52.5	0.5	-45	0.6	-62.5
5000000	0.5	-72.5	0.7	-27.5	0.6	-52.5	0.4	-17.5	0.4	-92.5

Withdraw

Training Steps	Agent 1		Agent 2		Agent 3		Agent 4		Agent 5	
	% perfect Games	Score	% perfect Games	Score	% perfect Games	Score	% perfect Games	Score	% perfect Games	Score
1000000	0	-37.5	0	-26.25	0	-37.5	0	-41.25	0	-37.5
2000000	0	-26.25	0	-37.5	0	-37.5	0	-37.5	0	-37.5
3000000	0	-32.5	0	-37.5	0.1	17.5	0	-52.5	0	-37.5
4000000	0.1	-28.75	0	-37.5	0.4	12.5	0	-37.5	0	-37.5
5000000	0.2	1.25	0.1	-12.5	0.4	61.25	0	-37.5	0	-37.5
6000000	0.1	16.25	0.1	-28.75	0.4	62.5	0	-37.5	0	-37.5
7000000	0.7	73.75	0.1	2.5	0.7	90	0	-37.5	0	-28.75
8000000	0.5	46.25	0.1	-15	0.6	82.5	0	-37.5	0	-22.5
9000000	0.4	45	0.3	13.75	0.6	62.5	0	-37.5	0	-26.25
10000000	0.8	81.25	0.1	2.5	0.7	87.5	0	-37.5	0	-23.75

B. TWO-VERSUS-ONE SPATIAL INVARIANCE

	Setup 1	Setup 2	Setup 3	Setup 4	Setup 5	Setup 6	Setup 7	Setup 8	Setup 9	Setup 10	Setup 11	Setup 12	Setup 13	Setup 14
Agent 1	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 1	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 1	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 1	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 1	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 1	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 1	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 1	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 1	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 1	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 1	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 2	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 2	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 2	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 2	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 2	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 2	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 2	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 2	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 2	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 2	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 3	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 3	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 3	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 3	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 3	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 3	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 3	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 3	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 3	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 3	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 4	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 4	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 4	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 4	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 4	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 4	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 4	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 4	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 4	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 5	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 5	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 5	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 5	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 5	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 5	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 5	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 5	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 5	0	100	0	100	0	100	0	0	100	0	100	0	100	0
Agent 5	0	100	0	100	0	100	0	0	100	0	100	0	100	0

C. TWO-VERSUS-TWO

Fixed-Positions

	Agent 1		Agent 2		Agent 3		Agent 4		Agent 5	
Training Steps	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score
100000	92.5	-37.5	99.75	-4	96.75	-22.75	80.625	-97.5	87.875	-48.75
200000	99.625	-1	120.625	-182	98.5	-10.25	82	-77.25	125.5	-230.3
300000	97.75	-19.5	130.625	-247.5	120.25	-233.8	97.125	-18.5	137.625	-205.8
400000	130.625	-164.8	137.125	-244	160.25	-81	152.75	-109.75	135.375	-205
500000	162.25	-80.75	145.875	-174.5	159.75	-75.5	160.5	-91.5	139.375	-206.5
600000	161.25	-82.5	147.5	-165	160.375	-80.75	162.25	-77.75	134.75	-201
700000	161.875	-76	147.5	-160.3	160.375	-76	161.5	-73.5	136.875	-201.8
800000	159	-72.25	150	-164.5	161.75	-73.5	153.375	-77.5	152.5	-104.5
900000	160.75	-73.25	145.875	-161.3	163	-76	161.5	-73	0	0
1000000	158	-80.75	149.875	-155.5	161.375	-75.75	161.75	-73.5	153.375	-113
1100000	162.25	-78	148.5	-166.3	162.75	-79.25	160.625	-74.5	153.75	-113.5
1200000	161.75	-76.75	149.125	-157.3	159.375	-75.5	163.75	-77.5	153	-103
1300000	164.25	-78.5	148.25	-155.5	162.75	-77.25	159.25	-78	153	-114.8
1400000	160.75	-91.25	150	-167	162.75	-75.5	163.25	-78.5	161.75	-77.25
1500000	156.5	-82	152.25	-162.5	163.5	-77	113.3125	-154.13	163	-76
1600000	163.25	-76.5	148	-165.3	163.25	-76.5	161.25	-74.25	163	-76
1700000	164	-78	149.75	-162.8	161.5	-81.75	161.25	-72.5	161.25	-72.5
1800000	161.25	-80.5	148.125	-164	160.5	-73.75	163.25	-80.75	114.5	-50.25
1900000	160.75	-71.5	161.5	-74.75	149.25	-88.5	161.75	-74.5	163.375	-79
2000000	160.25	-70.5	159.25	-76	162	-75.75	163.75	-77.5	162.625	-77.5

Multiple-Starting-Positions

	Agent 1		Agent 2		Agent 3		Agent 4		Agent 5	
Training Steps	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score
100000	75	-116.25	79.125	-95.5	84.375	-83.25	82	-197.75	71.375	-145
200000	112.875	-245	97.375	-29.75	98.375	-89.25	134.96875	-163.813	98.625	-13.5
300000	150.125	-112.5	150.78125	-124.438	152.90625	-94.4375	152.875	-101.25	106.75	-224.75
400000	142.1875	-132.375	151	-104.5	99.625	-204.5	154	-92.5	148.6875	-121.375
500000	153.53125	-78.6875	55.34375	-291.563	150.75	-96	159.25	-89.25	155	-81.25
600000	146.15625	-109.188	155.25	-87	95.5	-193.5	157.375	-86	123.25	-177.75
700000	131.125	-158.5	160	-82.5	161.875	-78.75	160.25	-71	157.125	-92
800000	154.53125	-92.6875	151.625	-97.75	161.875	-76.75	117.5	-62.75	141.75	-108.5
900000	134.8125	-102.625	159.75	-83.5	144	-89.5	144.875	-126.5	139.25	-105.5
1000000	162	-84.5	147.875	-96	162.53125	-83.1875	162	-76	159.125	-91
1100000	158	-84.75	160.75	-71.5	155.75	-91	162	-78	156.625	-81.75
1200000	145.8125	-106.125	158.375	-85.75	153	-84.5	148.03125	-112.688	162.875	-82.25
1300000	160.75	-80.25	158.25	-88.5	121.375	-106.25	159.875	-75.25	165	-80
1400000	160.5	-75	162.5	-77.75	142.75	-86.5	159.125	-79.75	162.125	-76.5
1500000	156.125	-89.25	79.5	-244.25	158.625	-80.5	88.5	-217.5	159.75	-79.75
1600000	162	-75	102.34375	-263.063	160	-73.5	161.125	-84.5	161.75	-75.5
1700000	159.875	-72	105.1875	-246.875	157	-86	161.875	-76.75	155.5	-93.25
1800000	158.53125	-81.6875	97.6875	-270.625	159.25	-84.75	161.75	-79.75	162.5	-75
1900000	161.25	-79	120.53125	-205.188	160.25	-72.5	163.5	-77	163.5	-77
2000000	163	-78	50.125	-81.75	162.25	-80.75	160.75	-78	163	-80.5
2100000	160.375	-76	92.375	-276.25	163.625	-87.25	3.25	-272	157.625	-77.75
2200000	162.5	-77.75	161	-79	156.125	-75.75	158.8125	-84.125	1	-0.5
2300000	162.875	-78.75	160.75	-84	160.9375	-74.625	163.25	-76.5	0	0
2400000	158.875	-81	160.375	-75.75	159.75	-69.5	160.5	-72.75	17	-36
2500000	161.75	-74	163	-76	163.25	-79.25	162.375	-86	63.5	-165.5
2600000	157.5	-76.75	75.40625	-48.6875	161.25	-72.5	163	-76	135.1875	-151.125
2700000	160.75	-71.5	159.5	-79	162.75	-75.5	148.40625	-104.438	162	-74
2800000	152	-100.5	160.75	-71.5	161.125	-75.25	162	-79.5	161.5	-75.5
2900000	162.25	-77	161.25	-72.5	163.25	-76.5	161.5	-73	161.125	-78.75
3000000	162.25	-74.5	163	-76	164	-78	160.875	-76.5	162.25	-79.75

D. THREE-VERSUS-TWO

Fixed Positions

	Agent 1		Agent 2		Agent 3		Agent 4		Agent 5	
Training Steps	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score
100000	98.5	-25.5	94.58333333	-42.5	97.75	-16	99.75	-2.75	74.5	-181.5
200000	99	-10	97.91666667	-17	97.83333333	-15.5	99.16666667	-9	74.66666667	-182.25
300000	100	-3	99.25	-10	99.75	-7.5	99.33333333	-4.5	74.5	-180.25
400000	99.66666667	-13.5	85.08333333	-98.75	98.41666667	-38.25	98.41666667	-8.75	75.16666667	-201
500000	100	-4.75	98.16666667	-17.25	98.16666667	-16.5	99	-9.25	108.5	-246.5
600000	97.66666667	-17.5	99.33333333	-6.75	100	-16	99.83333333	-8	113.6666667	-235.75
700000	97.33333333	-24.5	99.5	-3	99.58333333	-11.5	99.33333333	-6.75	122.25	-181
800000	100	0	100	-11	99.75	-5	98.91666667	-6.5	123.5	-183.5
900000	97.08333333	-23.75	98.41666667	-18.5	99.75	-1	99	-7.25	123	-183
1000000	98.41666667	-20.75	99.5	-7.75	100	-20.5	99	-4.25	119	-199.5
1100000	99.16666667	-9.5	99	-37.5	90.33333333	-92.5	98.66666667	-15.75	123.5	-181
1200000	99.33333333	-9.5	95.75	-38	140.75	-57.5	99.25	-5.75	94.25	-227.75
1300000	100	-1.75	99.66666667	-5.5	143.5	-39	91.08333333	-35.5	122.25	-187.5
1400000	100	-1	124.75	-31.5	132.4166667	-41	102.25	-14	149.5833333	-91.25
1500000	106.0833333	-14.25	130.75	-67.25	139.0833333	-37.25	104.75	-17.75	149.25	-90
1600000	131.3333333	-42	137.3333333	-62	139.1666667	-41.5	138.3333333	-48.5	152.5	-86.25
1700000	139.75	-28	139.75	-49	166.75	-8	137.75	-46.75	150.75	-88.75
1800000	147	-34.5	145.75	-35.25	170.5	-1.5	138.5	-34	151	-87
1900000	161.6666667	-13.25	138.25	-54	169.5	-2.5	140.5	-33.25	148.5	-91.75
2000000	170.25	-4.25	141.6666667	-22.5	171.75	-3.75	169	-2.75	167	-24.25
2100000	172	-3.25	131	-33.5	172	-2.25	168.25	-5.75	172.75	-6
2200000	172.75	-1.75	134.0833333	-14.25	174.25	-0.75	173.25	-1.75	173.25	-4
2300000	175	0	137.5	-5.75	174.75	-1	169	-7	172.5	-6.75
2400000	173.5	-2	138.25	-9.75	172.75	-1.75	164.5	-10.25	172.5	-11.25
2500000	172.5	-3.5	138.3333333	-14	168.25	-5.25	169.75	-4.5	170.5	-13
2600000	174.25	-0.75	137.75	-22.5	172.75	0	168.25	-6	172.3333333	-6.25
2700000	174.25	-0.75	137.9166667	-11	169.75	-3.25	171.25	-2.75	174.25	-2
2800000	174.25	-0.75	140.5	-0.5	173.25	-1.75	169.25	-6.75	165.75	-28
2900000	174	-4	144.6666667	-22	171.75	-5.5	169.75	-5.5	172.75	-6
3000000	175	0	131.8333333	-15.5	172.75	-1.5	172	-3.5	172.75	-4.75

Multiple Starting Positions

	Agent 1		Agent 2		Agent 3		Agent 4		Agent 5	
Training Steps	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score
100000	25	-176.5	95.75	-177	33.5	-132	93	-167.75	94.75	-176.7675
200000	101.8333333	-181	99.75	-175	66.75	-37.5	100.0833333	-189.25	94.16666667	-224.49
300000	168.4166667	-20	98	-179	92.41666667	-147.5	154.5	-130.5	173	-11.11
400000	171.4166667	-12.25	94.5	-136.5	167.75	-39.25	172.5833333	-17.25	171.3333333	-22.22
500000	173.5	-4.75	98.33333333	-180.25	172.1666667	-10.5	173.75	-8.75	165.8333333	-47.975
600000	174.25	-2.75	95.25	-185.75	169.75	-33.75	173.6666667	-9.25	169.4166667	-21.9675
700000	171.9166667	-12.25	95.16666667	-187.25	172.25	-14.75	174.3333333	-9	170.25	-21.21
800000	171.0833333	-12.25	96.75	-178	173.4166667	-12.5	174	-9.25	173	-10.3525
900000	174	-4.5	99.75	-175.25	168.5833333	-25.5	169.3333333	-30	139	-132.5925
1000000	174.8333333	-2	100	-175	173.5833333	-4.75	172.3333333	-19	173	-9.595
1100000	173.6666667	-5.5	125	-190.75	167.6666667	-22.5	174.75	-2.75	174.0833333	-5.555
1200000	173.3333333	-9.75	162.1666667	-99.75	169.8333333	-25.5	170.1666667	-17	173.3333333	-14.3925
1300000	171.9166667	-10.25	160.4166667	-101.5	171.9166667	-14.75	173.4166667	-11.5	172.8333333	-12.3725
1400000	175	-2.25	162.75	-84.25	160.9166667	-74.75	173	-10.25	173.75	-8.3325
1500000	172.5	-13.5	166.75	-54.75	171.75	-6.5	171	-23.75	174	-5.05
1600000	172	-11.75	173.9166667	-7.75	162.6666667	-71.75	173.3333333	-19.5	174.5833333	-4.04
1700000	166.4166667	-34.75	173.5	-8	174.5833333	-5.75	173.8333333	-13	174.5	-4.04
1800000	171.0833333	-16.75	173.4166667	-8	173.5833333	-8.25	160.8333333	-73.25	173.5	-2.7975
1900000	173.0833333	-13.25	172.25	-20.25	174.8333333	-3.25	57.75	-134.5	173.75	-6.8175
2000000	174.6666667	-2.5	173.6666667	-12.25	173.0833333	-5	174.8333333	-4.5	168.0833333	-21.4625
2100000	165.0833333	-25.25	173.9166667	-5.75	173.75	-12	165.0833333	-37.5	169	-17.9275
2200000	174.5	-2	174.5833333	-6.75	174.6666667	-2.5	175	-5.5	174.75	-1.01
2300000	173.25	-9.25	171.9166667	-17.5	173.25	-11.5	173.6666667	-9.75	170.5	-16.665
2400000	165.3333333	-28.25	174.3333333	-7.5	174.75	-1.5	168.8333333	-42.5	174.25	-3.7875
2500000	171	-8.25	174.8333333	-1	167.3333333	-39.75	174.75	-5	174.25	-3.2825
2600000	171.5	-17.25	174.75	-2.25	174	-3.5	175	-2.5	140	-128.27
2700000	172.5833333	-15.75	169.75	-33.25	166.6666667	-58.75	153.6666667	-69.75	174.25	-1.7675
2800000	173.25	-9.75	174.5	-3.25	173.3333333	-7.5	169.25	-33.5	175	-0.505
2900000	172.3333333	-6.5	175	-1.25	173.5	-9	174.0833333	-15.75	169.75	-19.9475
3000000	174.3333333	-3.5	165.25	-18.75	174.25	-5.5	174.8333333	-16.5	174.75	-4.2925
3100000	170.0833333	-12.75	174.75	-1	174.5	-3.5	166.25	-50.75	170.5	-19.19
3200000	174.8333333	-1.5	166.6666667	-25.5	174.5833333	-2	175	-6.5	174.25	-6.3175
3300000	167.0833333	-19.25	172.9166667	-18	173.0833333	-7.75	157.3333333	-73.25	173	-6.3125
3400000	171.6666667	-20	168.75	-25.75	154.5	-23	172.9166667	-12.25	170.3333333	-14.645
3500000	173.1666667	-6.5	175	0	173.9166667	-2.5	174.75	-4	174.4166667	-2.525
3600000	173.1666667	-8	172.25	-18.5	171.75	-23	174.0833333	-7.5	173.9166667	-4.545
3700000	174.6666667	-9.5	169.1666667	-28.75	172.9166667	-7.75	171	-25.5	172.75	-12.12
3800000	173.1666667	-26.5	173.75	-8.75	174.25	-11.5	174.75	-5.25	175	-5.555
3900000	174.8333333	-1.5	172.75	-9	173.1666667	-12	175	-15.75	175	-0.505
4000000	173.5	-7.75	173.4166667	-10.5	174.8333333	-6.5	174.8333333	-2	174.75	-0.505

E. URBAN TERRAIN

Urban Terrain Value 0

	Agent 1		Agent 2		Agent 3		Agent 4		Agent 5	
Training Steps	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score
100000	98.875	-7.25	98	-10	98.875	-7	100	0	100	0
200000	99	-4	99.25	-5	99.5	-3.5	98.125	-8.25	100	-0.5
300000	100	0	99	-4	100	0	98.875	-5.5	100	0
400000	100	0	90.25	-57.75	99.625	-2.25	99.25	-1.75	97.625	-7.75
500000	52.875	-116	97.25	-17.5	100	-1.25	100	-0.5	100	0
600000	99.25	-5	98.25	-9	99.75	-1	89.125	-54.25	94.25	-32.5
700000	100	0	99.5	-4.25	99.25	-3.75	99	-3.25	95.5	-22.25
800000	100	-0.5	100	0	100	0	100	0	100	0
900000	100	-0.75	96.25	-21	99.75	-1	100	0	96.5	-16.5
1000000	100	-1.25	99.625	-1.5	96.125	-22.75	99.25	-2.75	99.125	-4.5
1100000	99.25	-3.75	98.75	-8.25	100	0	99	-6.25	94.25	-33
1200000	99.25	-4.75	100	0	99.25	-4.75	97.875	-8.5	99.375	-2
1300000	100	0	100	-0.75	87.875	-64.25	99	-5.5	98.875	-5
1400000	100	0	100	-0.5	100	0	100	-1	100	0
1500000	100	0	93.125	-42.25	98.875	-5.5	99.5	-3.5	100	0
1600000	100	-0.5	100	0	100	-0.75	99.25	-3.75	99.25	-1.75
1700000	96.75	-14.75	99.25	-3.75	99.75	-1	100	-2	100	-0.5
1800000	100	0	99.75	-2	99.25	-4.25	99.5	-2	100	0
1900000	99.625	-1.75	100	-0.75	97	-17	100	0	100	0
2000000	100	0	100	0	100	0	100	0	100	0

Urban Terrain Value 20

	Agent 1		Agent 2		Agent 3		Agent 4		Agent 5	
Training Steps	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score
100000	218.4	118.4	157.2	58.8	180.85	77.35	99.125	-3.75	218.15	117.4
200000	230.05	115.85	158.8	78	240.8	113.8	174.975	73.6	232.65	109.9
300000	237.35	113.1	160	78.2	237.6	115.6	179.2	79.2	239.75	110.9
400000	238.9	112.6	156	80	240.4	114.9	170.825	52.65	239.2	115.2
500000	238.05	113.85	164.4	72.4	240	113.5	180	80	239.2	115.2
600000	238.4	115.4	159.2	80	237.6	115.6	178.45	75.45	245.2	113.7
700000	238.05	112.85	160.4	71.7	241.2	114.7	179.2	79.2	242	114.5
800000	238.4	113.9	160.8	80	239.6	115.1	179.6	79.6	238.4	113.1
900000	239.6	115.1	158.8	78.8	238	115.5	178.825	78.2	238.4	115.4
1000000	244	113.5	156.8	79.6	239.6	115.1	177.125	73	238	115.5
1100000	238	115.5	162	80	240	115	178	77.5	240.4	114.9
1200000	240	115	157.6	80	240.85	113.15	178.95	77.2	240.8	114.8
1300000	238.4	115.4	158.8	79.6	238.5	112.85	180	80	239.35	113.5
1400000	241.6	114.6	178.425	80	240.8	114.8	180.4	79.9	240.4	114.9
1500000	241.6	114.6	178.025	80	238.4	115.4	177.95	65.05	241.2	114.2
1600000	237.6	115.6	179.5	80	240	115	180	80	239.6	115.1
1700000	242.8	114.3	176.05	80	240	115	180	80	243.2	114.2
1800000	243.2	114.2	179.35	80	238	115.5	180	80	239.2	115.2
1900000	240.4	114.9	178.55	80	236	116	179.75	78	239.2	115.2
2000000	239.6	115.1	180	80	243.6	114.1	180	80	238.8	115.3

Urban Terrain Value 40

	Agent 1		Agent 2		Agent 3		Agent 4		Agent 5	
Training Steps	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score	Total Discounted Reward	Score
100000	586.8	585.8	586.8	586.8	596.4	596.4	585.25	586.25	485.3	485.3
200000	596.4	596.4	596.4	596.4	598.8	598.8	596.4	596.4	498.8	498.8
300000	597.6	597.6	597.6	597.6	600	600	600	600	500	500
400000	598.8	598.8	598.8	598.8	598.8	598.8	600	600	542.7	542.7
500000	600	600	600	600	600	600	600	600	600	600
600000	600	600	600	600	600	600	600	600	600	600
700000	600	600	600	600	600	600	600	600	600	600
800000	598.8	598.8	598.8	598.8	600	600	600	600	600	600
900000	600	600	600	600	600	600	598.8	598.8	600	600
1000000	600	600	600	600	600	600	600	600	600	600
1100000	600	600	600	600	600	600	600	600	600	600
1200000	600	600	600	600	600	600	597.6	597.6	600	600
1300000	600	600	600	600	600	600	600	600	600	600
1400000	600	600	600	600	600	600	600	600	600	600
1500000	600	600	600	600	600	600	600	600	598.8	598.8
1600000	600	600	600	600	600	600	600	600	600	600
1700000	600	600	600	600	600	600	594	594	600	600
1800000	600	600	600	600	600	600	600	600	600	600
1900000	600	600	600	600	600	600	600	600	600	600
2000000	600	600	600	600	600	600	600	600	600	600

F. MULTI-AGENT-TRAINING

Blue performance in complex setup (10 repetition)

	Blue Agent	
Training Steps	Reward	Score
1000000	380.375	315.35
2000000	399.8594	304.9094
3000000	495.05	449.05
4000000	494.9	395.375
5000000	569.2625	458.9125
6000000	496.9125	384.8875
7000000	572.8	535.3
8000000	438	383.7
9000000	403.65	298.15
10000000	591.25	500

G. LARGE-SCALE SCENARIO

Different Input Layers

Training	Standart		Unit Layers		Terrain Layers	
Steps	Reward	Score	Reward	Score	Reward	Score
50000	25.7198058	-758.877	0	-690	23.3288043	-572.5
100000	67.0820028	-1458.84	28.125	-785	0	-600
150000	231.276946	-2086.01	165.521744	-590	15	-632.5
200000	196.072486	-2166.61	220.692483	-520	13.75	-692.5
250000	264.312991	-1759.69	219.043631	-580	58.75	-687.5
300000	304.553839	-1883.72	219.043631	-1000	228.695867	-817.5
350000	293.231944	-1623.13	219.043631	-740	230.840506	-1437.19
400000	189.018528	-2198.22	219.043631	-780	44.0558511	-707.5
450000	204.467487	-2251.01	239.043631	-650.8	281.142992	-779.375
500000	188.467909	-1865.94	219.043631	-640	312.911326	-637.7
550000	241.137307	-1654.69	219.167194	-710	65.0587263	-610
600000	226.205485	-1551.21	219.043631	-720	219.367956	-565
650000	221.967277	-1535.55	219.043631	-680	219.530118	-557.5
700000	231.9116	-1615	219.043631	-620	309.058749	-395.8
750000	255.835588	-1560.63	219.043631	-680	268.905892	-561.963
800000	195.377071	-1136.41	215.368998	-573.75	319.05119	-324.9
850000	267.516954	-1652.03	224.043631	-685	319.058749	-466.6
900000	240.880728	-1404.69	233.816359	-665	161.152476	-603.3
950000	218.864251	-1391.88	250.026572	-610	319.061269	-379.1
1000000	233.339807	-1427.3	267.453617	-541.25	319.05371	-382.2
1050000	204.440082	-1244.69	324.600206	-458.75	267.166436	-625.663
1100000	191.697725	-1399.06	320.15678	-447.5	323.284377	-599.838
1150000	230.638432	-1260	323.589086	-499.9	358.475501	-357.2
1200000	188.874607	-1237.81	319.043631	-388.8	326.940985	-607.213
1250000	236.130714	-1333.13	279.043631	-704.5	359.788754	-357.9
1300000	244.282483	-1502.66	311.543631	-535.5	393.351438	-408.3
1350000	272.999633	-1677.66	307.596846	-450.1	404.562639	-453.138
1400000	251.626972	-1458.91	331.543631	-401.1	418.573809	-299.5
1450000	238.821008	-1162.89	321.786875	-411.1	418.763727	-269.5
1500000	251.281375	-1328.98	240.234905	-660.8	418.353782	-282.2
1550000	287.597869	-1543.13	307.139268	-414.9	359.592259	-592.838
1600000	339.590149	-1664.84	211.77155	-520.6	261.951933	-877.125
1650000	350.520434	-1488.59	0	-480	338.641232	-685
1700000	206.192767	-1461.04	0	-480	335.012405	-702.5
1750000	297.763594	-1669.38	123.315326	-690.313	307.448488	-673.425
1800000	351.962078	-1433.48	289.043631	-481.2	308.346545	-600.938
1850000	357.736374	-1322.23	339.043631	-388	210.45277	-667.675
1900000	356.688509	-1537.23	331.543631	-390.7	394.037767	-416.9
1950000	240.841815	-1942.87	344.043631	-359	397.887745	-366.7
2000000	368.979912	-1513.44	344.043631	-359	349.373572	-581
2050000	332.547977	-1631.36	299.731491	-475.3	394.515133	-492.313
2100000	340.899863	-1550.78	336.543631	-426.5	331.258053	-444.3
2150000	346.934973	-1461.09	152.355763	-1033.44	418.811816	-254.5
2200000	371.164771	-1527.34	324.043631	-446.2	250.271674	-680.408
2250000	360.581356	-1398.91	341.543631	-361.5	418.775928	-261.8
2300000	352.796811	-1385.63	344.043631	-359	356.064569	-863.175
2350000	343.960811	-1455	341.543631	-457.5	363.56075	-697.6
2400000	348.475469	-1470.63	331.543631	-467.5	95.7118882	-848.75
2450000	338.409107	-1570.78	304.639268	-462.5	141.937983	-765
2500000	296.426394	-1704.79	316.543631	-482.5	327.788462	-824.213
2550000	385.118886	-1455.16	339.043631	-460	310.517198	-722.45
2600000	379.757016	-1331.25	311.543631	-487.5	367.444762	-785.3
2650000	338.922441	-1498.71	339.043631	-460	365.938097	-705.1
2700000	362.916117	-1277.66	329.043631	-470	413.24266	-622.5
2750000	365.036163	-1442.66	311.543631	-487.5	281.127356	-758.425
2800000	367.541673	-1375.47	68.8087263	-635	330.502425	-773.938
2850000	373.565682	-1346.88	0	-1140	353.427637	-602.55
2900000	343.42124	-1410.16	148.550974	-1500.31	404.212193	-575.2
2950000	377.432847	-1387.66	187.536102	-1447.21	308.640424	-701.25
3000000	367.911695	-1330.04	272.907637	-1309.38	413.449835	-597.5

MLP/ CNN

Training Steps	MLP		CNN	
	Reward	Score	Reward	Score
1000000	584.286914	-199.388	237.57487	-1284.38
2000000	775.636353	-250.593	265.922038	-1330.83
3000000	787.578446	-213.474	314.437109	-1444.77
4000000	785.254675	-177.587	288.128874	-1676.06
5000000	774.320915	-225.487	356.492155	-1466.53
6000000	790.564478	-166.618	135.20682	-2273.04
7000000	785.122811	-202.611	221.153361	-1602.27
8000000	740.768164	-79.15	247.592773	-1516.88
9000000	752.968164	-79.55	265.680237	-1563.21
10000000	750.576497	-41.325	248.955078	-1471.41

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] R. Smith, "The long history of gaming in military training," *Simul. Gaming*, vol. 41, no. 1, pp. 6–19, 2010. [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/a550307.pdf

[2] E. H. Page and R. Smith, "Introduction to military training simulation: a guide for discrete event simulationists," in *1998 Winter Simulation Conference. Proceedings (Cat. No.98CH36274)*, Dec. 1998, vol. 1, pp. 53–60 vol.1, doi: 10.1109/WSC.1998.744899.

[3] D. H. Andrews, T. Dineen, and H. H. Bell, "The use of constructive modeling and virtual simulation in large-scale team training: A military case study," *Educ. Technol.*, vol. 39, no. 1, pp. 24–28, 1999.

[4] S. Yildirim and S. B. Stene, "A survey on the need and use of AI in game agents," *Model. Simul. Optim. - Focus Appl.*, Mar. 2010, doi: 10.5772/8968.

[5] O. Vinyals *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, Nov. 2019, doi: 10.1038/s41586-019-1724-z.

[6] J. Boron, "Developing combat behavior through reinforcement learning," Master M.S. Thesis, Dept. of Comp. Sci, NPS, Monterey, CA, USA, 2020. [Online]. Available: https://calhoun.nps.edu/bitstream/handle/10945/65414/20Jun_Boron_Jonathan.pdf?sequence=1&isAllowed=y

[7] Y. Sun, B. Yuan, T. Zhang, B. Tang, W. Zheng, and X. Zhou, "Research and implementation of intelligent decision based on a priori knowledge and DQN algorithms in wargame environment," *Electronics*, vol. 9, no. 10, p. 1668, Oct. 2020, doi: 10.3390/electronics9101668.

[8] W. M. Lian, "Using neural networks to determine course of action for a land-based constructive simulation," Dept. of Comp. Sci, NPS, Monterey, CA, USA, 2020. [Online]. Available: https://calhoun.nps.edu/bitstream/handle/10945/63476/19Sep_Lian_Weiwen%20Mervyn.pdf?sequence=1&isAllowed=y

[9] Thomas Mitchell, *Machine Learning*, 1st ed. McGraw-Hill, Inc., 1997.

[10] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ: Prentice Hall Press, 2009.

[11] V. Francois-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, “An introduction to deep reinforcement learning,” *Found. Trends® Mach. Learn.*, vol. 11, no. 3–4, pp. 219–354, 2018, doi: 10.1561/2200000071.

[12] H. Wang *et al.*, “Integrating reinforcement learning with multi-agent techniques for adaptive service composition,” *ACM Trans. Auton. Adapt. Syst.*, vol. 12, pp. 1–42, May 2017, doi: 10.1145/3058592.

[13] L. Engstrom *et al.*, “Implementation matters in deep policy gradients: A case study on PPO and TRPO,” *ArXiv200512729 Cs Stat*, May 2020, Accessed: Oct. 21, 2020. [Online]. Available: http://arxiv.org/abs/2005.12729.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *ArXiv170706347 Cs*, Aug. 2017, Accessed: Oct. 21, 2020. [Online]. Available: http://arxiv.org/abs/1707.06347.

[15] OpenAI, “Proximy policy optimization,” OpenAI Spinning Up, 2018. Accessed Mar. 23, 2021 [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/ppo.html

[16] M. Mahmud, M. S. Kaiser, A. Hussain, and S. Vassanelli, “Applications of deep learning and reinforcement learning to biological data,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2063–2079, Jun. 2018, doi: 10.1109/TNNLS.2018.2790388.

[17] OpenAI, “OpenAI Five defeats Dota 2 world champions,” *OpenAI*, Apr. 15, 2019. Accessed Oct. 15, 2020 [Online]. Available: https://openai.com/blog/openai-five-defeats-dota-2-world-champions/#fn2

[18] D. Mwiti, “10 real-life applications of reinforcement learning,” *neptune.ai*, Jul. 22, 2020. Accessed Oct. 21, 2020. [Online]. Available: https://neptune.ai/blog/reinforcement-learning-applications

[19] L. Hardesty, “Explained: Neural networks - Ballyhooed artificial-intelligence technique known as ‘deep learning’ revives 70-year-old idea.,” *MIT News on Campus and around the world*, Apr. 14, 2017. Accessed Oct. 14, 2020. [Online]. Available: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414

[20] G. Allen, “Understanding AI technology. A concise, practical, and readable overview of artificial Intelligence and machine Learning technology designed for non-technical managers, officers, and executives.” Joint Artificial Intelligence Center (JAIC), Apr. 2020, Accessed: Oct. 14, 2020. [Online]. Available: https://www.ai.mil/docs/Understanding%20AI%20Technology.pdf

[21] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*, 4th ed. Cambridge University Press, 2003.

[22] F. Bre, J. M. Gimenez, and V. D. Fachinotti, "Prediction of wind pressure coefficients on building surfaces using artificial neural networks," *Energy Build.*, vol. 158, pp. 1429–1441, Jan. 2018, doi: 10.1016/j.enbuild.2017.11.045.

[23] A. Shertinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *Phys. Nonlinear Phenom.*, vol. 404, Mar. 2020.

[24] M. S. Researcher PhD, "simple introduction to convolutional neural networks," *Medium*, Jul. 29, 2020. Accessed Mar. 05, 2021. [Online]. Available: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

[25] "CS231n convolutional neural networks for visual recognition." Accessed Oct. 21, 2020. [Online]. Available: https://cs231n.github.io/convolutional-networks/

[26] M. Mishra, "Convolutional neural networks, explained," *Medium*, Sep. 02, 2020. Accessed Mar. 05, 2021. [Online]. Available: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

[27] M. Egorov, "Multi-agent deep reinforcement learning," 2016. Accessed Nov. 27, 2020. [Online]. Available: http://cs231n.stanford.edu/reports/2016/pdfs/122_Report.pdf

[28] M. C. Fu, "AlphaGo and Monte Carlo tree search: The simulation optimization perspective," in *2016 Winter Simulation Conference (WSC)*, Dec. 2016, pp. 659–670, doi: 10.1109/WSC.2016.7822130.

[29] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition," *ArXiv171003748 Cs*, Mar. 2018, Accessed: Nov. 13, 2020. [Online]. Available: http://arxiv.org/abs/1710.03748.

[30] Feng-Hsiung Hsu, "IBM's Deep Blue Chess grandmaster chips," *IEEE Micro*, vol. 19, no. 2, pp. 70–81, Mar. 1999, doi: 10.1109/40.755469.

[31] N. Brown and T. Sandholm, "Superhuman AI for heads-up no-limit poker: Libratus beats top professionals," *Science*, vol. 359, no. 6374, pp. 418–424, Jan. 2018, doi: 10.1126/science.aao1733.

[32] OpenAI *et al.*, "Dota 2 with large scale deep reinforcement learning," *ArXiv191206680 Cs Stat*, Dec. 2019, Accessed: Oct. 15, 2020. [Online]. Available: http://arxiv.org/abs/1912.06680.

[33] L. Stone, “AI pilot beats human 5:0 in DARPA dogfight,” *AI BUSINESS*, Aug. 24, 2020. Accessed Mar. 16, 2021. [Online]. Available: https://aibusiness.com/document.asp?doc_id=763402

[34] G. Moy and S. Shekh, “The Application of AlphaZero to Wargaming,” in *AI 2019: Advances in Artificial Intelligence*, vol. 11919, J. Liu and J. Bailey, Eds. Cham: Springer International Publishing, 2019, pp. 3–14.

[35] A. Choudhary, “A hands-on introduction to Deep Q-Learning using OpenAI Gym in Python,” *Analytics Vidhya*, Apr. 18, 2019. Accessed: Oct. 20, 2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

[36] “SVG Tutorial.” Accessed Mar. 18, 2021. [Online]. Available: https://www.w3schools.com/graphics/svg_intro.asp.

[37] A. Raffin, “Stable Baselines: a fork of OpenAI Baselines — reinforcement learning made easy,” *Medium*, May 08, 2020. Accessed Nov. 06, 2020. [Online]. Available: https://towardsdatascience.com/stable-baselines-a-fork-of-openai-baselines-reinforcement-learning-made-easy-df87c4b2fc82

[38] “Welcome to Stable Baselines3 docs! - RL Baselines made easy — Stable Baselines3 0.11.0a0 documentation.” Accessed Nov. 06, 2020. [Online]. Available: https://stable-baselines3.readthedocs.io/en/master/

[39] OpenAI, “Gym: A toolkit for developing and comparing reinforcement learning algorithms.” Accessed Nov. 06, 2020. [Online]. Available: https://gym.openai.com

[40] A. Patel, “Hexagonal grids,” *Red Blob Games*, May 2020. Accessed Apr. 14, 2021. [Online]. Available: https://www.redblobgames.com/grids/hexagons/

[41] AurelianTactics, “PPO hyperparameters and ranges,” *Medium*, Jul. 28, 2018. Accessed Mar. 18, 2021. [Online]. Available: https://medium.com/aureliantactics/ppo-hyperparameters-and-ranges-6fc2d29bccbe

[42] C. Steppa and T. L. Holch, “HexagDLy—Processing hexagonally sampled data with CNNs in PyTorch,” *SoftwareX*, vol. 9, pp. 193–198, Jan. 2019, doi: 10.1016/j.softx.2019.02.010.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California